UNIVERSIDAD CARLOS III DE MADRID

Departamento de Mecánica de Medios Continuos y Teoría de Estructuras

Grado en Ingeniería Tecnologías Industriales

Trabajo Fin de Grado

Análisis del Fallo en Componentes Estructurales Metálicos Sometidos a Procesos de Perforación

Alumno:
Víctor Antona García

Tutor:
Marcos Rodríguez MIllán

Julio 2014

Página en blanco intencionadamente

Índice general

Índice de figuras

Índice de tablas

Índice de gráficas

Resumen

El presente proyecto realiza una comparación de los resultados experimentales y numéricos obtenidos al estudiar el comportamiento de una placa de 4mm de aleación de aluminio AA 5754-H111, al ser perforada por proyectiles de cabeza plana, esférica y cónica.

En primer lugar, se han tomado como referencia para los ensayos experimentales los resultados publicados en el artículo científico "Experimental Study on the Perforation Process of 5754-H111 and 6082-T6 Aluminium Plates Subjected to Normal Impact by Conical, Hemisphericaland Blunt Project".

El análisis numérico de los resultados experimentales se ha realizado mediante el código comercial de elementos finitos ABAQUS. Para ello, se ha desarrollado un script, de manera que se pueda automatizar la interacción entre el usuario y ABAQUS.

Por último, se ha analizado y comparado con éxito los resultados experimentales con los numéricos, haciendo especial hincapié en las velocidades residuales del proyectil, así como en la respuesta a perforación, deformación y la energía absorbida por las placas para velocidades del proyectil que van desde los 120 m/s hasta los 200 m/s.

1.Introducción

1.1. Motivación

El análisis de impactos, así como el uso de los programas de Elementos Finitos, son dos campos de vital importancia en la actualidad para el diseño y desarrollo de componentes en la mayoría de los campos de la industria.

La necesidad de diseñar nuevas estructuras protectoras que sean capaces de soportar impactos sin que éstas sufran daños, es un problema que atañe prácticamente a todos los ámbitos ingenieriles, desde los sectores de vehículos y transportes, hasta los relacionados con seguridad y defensa militar, pasando por los campos de la aeronáutica o de la ingeniería naval. Uno de los aspectos clave para el diseño de dichas estructuras es la elección del material, por ello el estudio de su comportamiento al impacto (variando geometría del proyectil, velocidades y espesores de la placa) puede ser de gran utilidad. Por motivos de peso, prestaciones y contaminación, las aleaciones no ferríticos han suplantado a los materiales de base ferrosa en la industria. Concretamente las aleaciones alumínicas están enormemente extendidas en el panorama ingenieril, por lo que conocer su comportamiento mecánico y su capacidad de absorción de energía, podría permitir seguir mejorando las prestaciones de los materiales con el fin de conseguir una mayor relación entre su peso y su resistencia.

Por otro lado, los ensayos experimentales, además del elevado coste que pueden suponer, no siempre pueden darnos la información solicitada. Por tanto, es necesaria la utilización de modelos de simulación de Elementos Finitos que puedan predecir los resultados experimentales, reduciendo los costes y los tiempos de diseño. Así, la nueva interfaz de ABAQUS, Python ABAQUS Scripting, permite crear modelos de simulación de Elementos Finitos mediante líneas de código en lenguaje de programación Python, lo que reduce el tamaño de los archivos de los proyectos realizados y el tiempo a la hora de realizar cambios sobre el modelo.

1.2. Objetivos

- Aprendizaje del diseño modelos de simulación de Elementos finitos mediante Python ABAQUS Scripting
- Realización de simulaciones de impacto en placas de metal con diferentes geometrías de proyectil, dando lugar a diferentes mecanismos de fallo del componente estructural
- Obtención de una herramienta numérica capaz de reproducir ensayos de impacto en placas metálicas.

1.3. Estructura del documento

Antes de continuar desarrollando el trabajo realizado para este proyecto, se va a explicar brevemente el contenido general del mismo, así como el orden que se ha ido siguiendo.

Los apartados que se van a ir siguiendo serán los siguientes:

- **Materiales:** En este epígrafe se explicarán las características (densidad, módulo de Young, etc.) de los materiales con los que se ha trabajado en las distintas simulaciones.

- **Desarrollo de los Modelos de Simulación:** A lo largo de este capítulo se darán los detalles acerca de cómo se ha creado el modelo de impacto mediante Abaqus Scripting, haciendo principal hincapié en el código utilizado. Por otro lado, se comentarán las distintas simulaciones analizadas (cambios de geometría, de material, de proyectil, etc.)

- **Comparación de Resultados:** En este apartado se compararán los resultados obtenidos en las simulaciones con los datos experimentales sacados del artículo "*Experimental Study on the Perforation Process of 5754-H111 and 6082-T6 Aluminium Plates Subjected to Normal Impact by Conical, Hemisphericaland Blunt Project*".

- **Conclusiones**

2. Materiales

Resumen del capítulo

A lo largo de este capítulo se explicará de manera amplia y general las características y usos de las aleaciones de aluminio. Además, se profundizará en las características de la aleación utilizada como material de la placa durante las simulaciones de impacto: AA 5754-H111.

Se ha decidido llevar a cabo el estudio del impacto en dicha aleaciones debido a su habitual uso en casi todos los ámbitos de la industria ingenieril.

2.1. Características generales de las aleaciones de aluminio

Las aleaciones de aluminio han ido aumentando su importancia en la industria exponencialmente desde la década de los 40. Actualmente, es uno de los metales de mayor uso y relevancia en industrias como la aeronáutica o la del automóvil. Este gran interés, ha supuesto que sea necesario el estudio de su comportamiento y propiedades mecánicas.

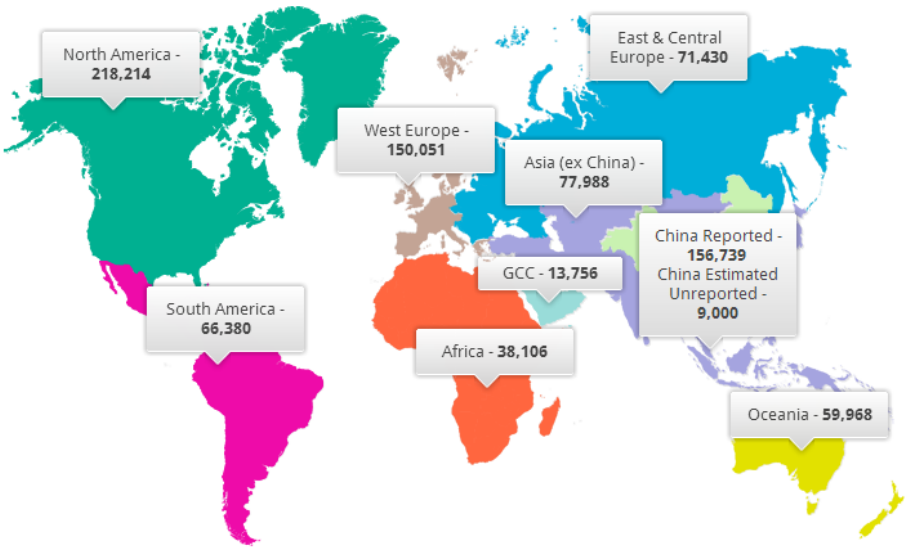


Figura 2. 1: Cantidad de toneladas producidas desde 1973 hasta la actualidad [1].

Sin embargo, a pesar de que las propiedades de las aleaciones de aluminio tienen ciertos parecidos con las del acero estructural (otro de los metales más utilizados en la industria), las de este último suelen estar por encima, sobre todo en términos de ductilidad.

La principal diferencia que ha suscitado tanto interés en el uso de dichas aleaciones es la diferencia de peso, mucho menor, respecto a las prestaciones que es capaz de proporcionar, en comparación con la mayoría de metales habitualmente utilizados en los sectores industriales, incluyendo el acero. Esto supone un gran aliciente para la investigación por parte de las industrias del transporte, ya que al reducir el peso de las estructuras se podrá ahorrar en el consumo de combustible, y por tanto en las emisiones de CO_2 y demás agentes contaminantes [2].

2.1.1. Ventajas y desventajas de las aleaciones de aluminio

Las ventajas principales de las aleaciones de aluminio:

- Poco peso.
- Permite ser mecanizado o extruido de forma sencilla.
- Se le pueden añadir capas de adhesivos.
- Inoxidable.
- Trabajan bien a temperaturas bajas.
- Se suelda con facilidad.

Sin embargo, también cuenta con una serie de desventajas, sobre todo en comparación con el acero, que serán aquí mencionadas:

- Los costes de fabricación son un 50% más alto por m^3 que los del acero. Pudiendo ser aún mayores en aleaciones de alta resistencia, como las utilizadas en el ámbito aeronáutico.
- Menos resistencia al pandeo que del acero.

- Aunque tiene buenas propiedades elásticas, son menores que las del acero y esto conlleva un alto índice de deflexión.
- Se comporta peor a altas temperaturas que el acero, pudiendo expandirse o contraerse hasta el doble de lo que lo hace un acero.
- Al interaccionar con otros metales, puede sufrir corrosión electrolítica.
- Problemas de fatiga.

2.1.2. Clasificación

Las aleaciones de aluminio según su estructura atómica, son clasificadas como estructuras cúbicas centradas en las caras (FCC), en las cuales por lo general, existen multitud de planos de deslizamiento [3]. Sin embargo, las propiedades de cada aleación, van a variar cuantiosamente en función de los elementos a los que esté aleado el Aluminio (Cobre, Manganeso, Silicio o Zinc entre otros) y de los tratamientos a los que se someta la aleación, entre los que destacan: endurecimiento por deformación y tratamientos de temperatura [3].

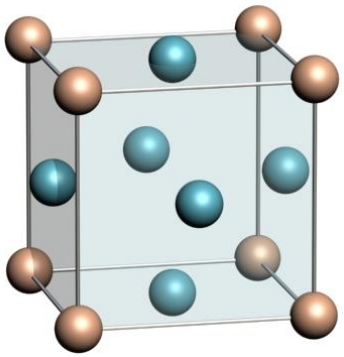


Figura 2. 2: Estructura cristalina del aluminio

La clasificación de las aleaciones de aluminio se ha estandarizado, utilizándose en todo el mundo el sistema americano. Este sistema denomina a cada aleación con dos secuencias, la primera de cuatro números y la segunda de números y letras, separadas por un guión. Como por ejemplo 5754-H111 ó 5083-O.

Los cuatro dígitos de la primera secuencia de números tienen el siguiente significado:

- **Primer dígito:** Indica el tipo de aleación, que dependerá de su aleante principal. Como se muestra en la tabla adjunta.
- **Segundo dígito:** Indica el límite de impurezas o si la aleación original ha sido cambiada.
- **Tercer y cuarto dígito:** Indican la pureza en aluminio de la aleación.

Nomenclatura	Principal elemento aleante	Tipo
1xxx	Aluminio puro (Al)	NHT
2xxx	Cobre (Cu)	HT
3xxx	Manganeso (Mn)	NHT
4xxx	Silicio (Sl)	NHT
5xxx	Magnesio (Mg)	NHT
6xxx	Magnesio (Mg) y Silicio (Sl)	HT
7xxx	Zinc (Zn) y Magensio (Mg)	HT
8xxx	Otros elementos	-
9xxx	Series no utilizadas	-

Tabla 2. 1: Código del aluminio en función de su aleante principal

La secuencia de letras y números que prosigue a la primera, revela los tratamientos de endurecimiento a los que ha sido sometida la aleación en cuestión. Las aleaciones de aluminios se pueden diferenciar en cuatro tipos según los tratamientos a las que hayan sido expuestas: aquellas que

pueden ser tratadas térmicamente y las que no, las que han sido sometidas a un recocido y las que son usadas sin tratamientos térmicos posteriores.

Las primeras, se designan con la letra H y cuentan con una amplia gama de posibilidades con las que ser tratadas o modificadas, tal y como se muestra en la siguiente tabla.

Código	Tratamientos
T1	Enfriamiento desde el conformado a temperatura elevada y maduración natural.
T2	Enfriamiento desde el conformado a temperatura elevada, deformación y maduración natural
T3	Tratamiento térmico de solución, deformación y maduración natural.
T4	Tratamiento térmico de solución y maduración natural.
T5	Enfriamiento desde el conformado a temperatura elevada y maduración artificial
T6	Tratamiento térmico de solución y maduración artificial
T7	Tratamiento térmico de solución y estabilización
T8	Tratamiento térmico de solución, deformación y maduración artificial.
T9	Tratamiento térmico de solución, maduración natural y deformación
T10	Enfriamiento desde el conformado a temperatura elevada, deformación y maduración artificial

Tabla 2. 2: Código del aluminio en función del tratamiento térmico

Por otro lado, el segundo tipo de aleaciones se indica con la letra T y son endurecidas mediante deformación en frío. Dependiendo del nivel de resistencia y dureza que se quiera alcanzar, la temperatura durante el proceso variará. Pudiéndose lograr un endurecimiento a un 25%, a un 50%, a un 75% o completo. El dígito inmediatamente posterior a la letra T, indica el proceso empleado para obtener el nivel de dureza conseguido, tal y como se muestra en la siguiente tabla.

Código	Tratamientos
H1	Endurecimiento por deformación en frío.
H2	Endurecimiento por deformación en frío y recocido parcial.
H3	Endurecimiento por deformación en frío y estabilización de las propiedades mecánicas por un tratamiento a baja temperatura

Tabla 2. 3: Código del aluminio en función del tratamiento de deformación

Las aleaciones denominadas con la letra O, son tratadas con un recocido y luego recristaliza totalmente gracias a un tratamiento térmico que, aunque reduce su resistencia, aumenta su ductilidad.

Por último, las aleaciones que no reciben ningún tratamiento posterior a su fabricación en caliente, son definidas por la letra F. Las propiedades de estos materiales no se pueden precisar de la misma manera que las otras, ya que no llevan un control tan riguroso.

Gracias a esta clasificación, podemos identificar las características principales de la aleación utilizada durante este estudio. Más en adelante se describirá en profundidad sus propiedades más importantes.

2.2. Usos habituales

El continuo aumento del uso del Aluminio en la mayoría de las industrias, ha convertido este material en casi imprescindible. La posibilidad de conseguir aleaciones con diferentes propiedades mecánicas, en función de nuestras necesidades, ha hecho que el Aluminio haya invadido el mercado de prácticamente todas las industrias. Sus usos son tantos, que aquí solamente se mencionarán los sectores de mayor relevancia y se explicarán brevemente en cada uno [4]

2.2.1. Electricidad y comunicación

El aluminio es una de las maneras más económicas de transportar electricidad de una manera eficiente. Con el paso del tiempo, ha ido sustituyendo al Cobre en las líneas de alto voltaje, siendo capaz de conducir electricidad a más de 700.000 V. Además también es utilizado en las antenas de televisión.

2.2.2. Transporte

En los últimos diez años, el uso de las aleaciones de aluminio ha aumentado de manera lineal en la principal industria del transporte, la automovilística.

Esto no se debe exclusivamente a motivos económicos, sino también ecológicos. En la actualidad existen coches fabricados íntegramente de aluminio, lo que conlleva reducciones del peso de hasta el 30%, con su consecuente ahorro de combustible y emisión. Además en zonas como América del Norte y Europa, se consigue reciclar más del 98% del aluminio utilizado en la construcción de automóviles.

Su ligereza es una ventaja en todos los sectores del transporte, siendo también una pieza fundamental en la construcción de ferrocarriles o aviones. Por ejemplo en el sector ferroviario, durante la vida media de un

tren construido de aluminio, se puede ahorrar hasta un 87% de energía en comparación con uno de otros materiales más pesados.

Figura 2. 3: Coche y avión fabricados de aluminio

2.2.3. Construcción

En Europa ha crecido enormemente la utilización del aluminio en construcciones y edificaciones en los últimos 50 años. Además de su habitual uso en las estructuras de ventanas o puertas, cada vez es más habitual ver este material como elemento principal de grandes cubiertas o en elementos ornamentales de edificios como se puede observar en las fotografías.

Figura 2. 4: "Ciudade las Artes y las Ciencias", Valencia, España y "The Wave", Almere, Holanda

2.2.4. Envasado

Es un campo en el que quizá resulte extraño encontrar al aluminio como material de importancia, sin embargo, tiene grandes ventajas frente a los envases de otros metales.

- Es 100% reciclable
- Protege el contenido contra la luz y el oxígeno durante mucho tiempo
- Son ligeros y difíciles de romper
- Enfrían el contenido rápidamente

Figura 2. 5: Embases fabricados de aluminio**.**

2.3. Aleación de aluminio AA 5754-H111

La aleación de aluminio utilizada para todos los ensayos de este estudio ha sido la AA 5754-H111, que como se ha explicado en un epígrafe anterior, es una aleación formada principalmente por Aluminio (Al) y Magnesio (Mg) (5xxx) y no puede ser tratada térmicamente (H). Además el código H111 nos indica que ha sido sometida a una deformación en frío tras su conformado, para mejorar sus propiedades mecánicas.

El porcentaje en peso de los aleantes que forman la aleación se muestra en la siguiente tabla:

Elemento	Mg	Fe	Sl	Mn	Cu	Cr	Tl	Zn
%	2.800	0.320	0.290	0.260	0.040	0.030	0.030	0.020

Tabla 2. 4: Tabla de composición del Aluminio AA- 5754-H111

A parte de la información que podemos obtener por su nomenclatura y de la tabla de composición, es relevante conocer otras propiedades de esta aleación. Conserva bastante bien sus características a bajas temperaturas y se puede soldar con sencillez. Además, tiene una gran resistencia a la corrosión, lo que la convierte en un material ideal para proyectos de estructuras que vayan a estar en contacto con el mar o en ambientes altamente contaminados.

Todas estas características le hacen que sea muy habitual su utilización en la industria aeronáutica, estructuras de almacenaje y válvulas de presión, centrales nucleares, además de complejos que trabajen a temperaturas del orden de 3 Kº. También es habitual su uso en ciertos componentes de los coches.

Su gran presencia en las industrias de transporte, hacen que sea necesario el estudio de su comportamiento a impacto.

3. Ensayos Experimentales

Resumen del Capítulo

Los datos experimentales han sido tomados del artículo “Experimental Study on the Perforation Process of 5754-H111 and 6082-T6 Aluminium Plates Subjected to Normal Impact by Conical, Hemisphericaland Blunt Project” realizado por M. Rodríguez-Millán, A. Vaz-Romero, A. Rusinek, J.A. Rodríguez-Martínez y A. Arias. El cual, tienen como objetivo determinar las características mecánicas de las aleaciones de aluminio previamente descritas, así como su comportamiento frente al impacto de proyectiles a distintas velocidades [5].

3.1. Descripción de los ensayos

Los ensayos se realizaron sobre placas de dimensiones de 130 mm x 130 mm y 4 mm de grosor de aleación de Aluminio tipo AA 5754-H111.

Dichas placas, fueron sujetas mediante un soporte rígido impidiendo cualquier movimiento, deslizamiento o vibración cuando recibieran el impacto.

Los proyectiles pretendían ser lo más parecido a un sólido indeformable, para lo cual se utilizaron proyectiles de Acero de con un alto Límite Elástico ($\sigma_y \approx 2000$ MP), con un tratamiento superficial especial para aumentar aún más su dureza y una masa de 30 g.

Los proyectiles tienen la siguiente geometría:

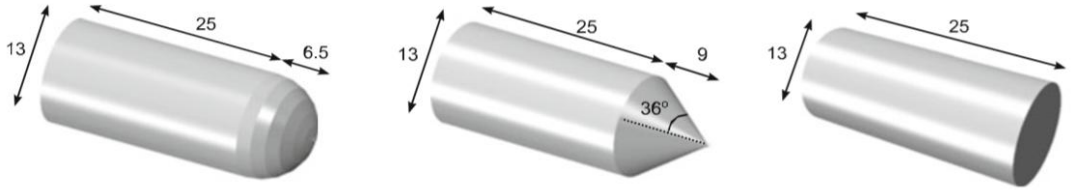


Figura 3. 1: Geometría y dimensiones de los proyectiles de los ensayos en mm

El impacto del proyectil se realiza sobre el centro de la placa y de manera perpendicular a ésta. Para garantizar que la trayectoria del proyectil formara 90º con la placa, se utilizó un cañón de aire comprimido convenientemente orientado con un cañón de diámetro muy similar al de los proyectiles, para evitar que la holgura pudiera variar el ángulo de incidencia.

Por último, para medir tanto las velocidades iniciales como las residuales del proyectil, se utilizó un sensor láser de fotodiodos con un contador de tiempo, que realizaba dos mediciones antes del impacto y otras dos después del mismo, calculando las velocidades inicial y residual, como la media entre ambos valores.

Para estudiar el comportamiento de la placa, se realizaron numerosos impactos con un rango de velocidades entre 120 m/s y 200 m/s con los tres proyectiles descritos y con el montaje indicado.

3.2. Conclusiones del ensayo experimental

Realizando estos ensayos se pretende cuantificar la influencia de las características del material, de la geometría del proyectil y de la velocidad sobre las perforaciones. De estos análisis se obtienen dos ideas innovadoras y muy importantes para el estudio de impactos. La primera, es que la eficiencia de la penetración para una geometría dada del proyectil, depende del material de la placa. La segunda, es que la cantidad de energía absorbida por la placa de un determinado material, depende de la velocidad y la geometría del proyectil.

4. Desarrollo de los Modelos de Simulación

Resumen del Capítulo

A lo largo de este capítulo se explicarán todos los aspectos necesarios para comprender el uso de Python ABAQUS Scripting, desde las características del lenguaje de programación hasta el funcionamiento y ventajas del programa.

Por otro lado, se describirá paso a paso el diseño del modelo, así como el código utilizado para llevarlo a cabo.

4.1. Descripción del Método de Elementos Finitos

Para poder utilizar de manera correcta los programas de simulación que trabajan con el Método de Elementos Finitos (MEF), debemos comprender primero como funciona dicho método de cálculo para una mayor comprensión de los resultados obtenidos.

El MEF es un método numérico utilizado para describir el comportamiento mecánico de elementos o estructuras que se basa en la simplificación de las ecuaciones diferenciales que rigen dicho comportamiento, en otras ecuaciones algebraicas más sencillas.

Para ello simplificamos la compleja geometría del modelo en cuestión en partes o elementos de geometría más sencilla que nos faciliten los cálculos, basándonos en el Principio de la Energía Potencial Mínima.

Cada elemento finito que hemos definido, tiene unos nodos característicos que nos permiten evaluar las variables de desplazamientos, deformaciones y tensiones en su interior, gracias al conjunto de funciones de interpolación local correspondiente a cada elemento.

A continuación, integrando el volumen del elemento hallamos la matriz de rigidez local. Posteriormente ensamblamos todas las matrices locales,

correspondiente a cada uno de los elementos finitos, construyendo la matriz de rigidez global del modelo.

Este método de cálculo que se ha descrito aquí brevemente, se ha conseguido implementar en programas informáticos como Abaqus. Esto es un importante avance en el campo del diseño de estructuras o componentes y en el desarrollo y análisis de prototipos, ya que nos permite obtener resultados aproximados de las solicitaciones de estructuras complejas sin necesidad de hacer ensayos experimentales, lo que supondría un gasto económico que no es asumible.

4.2. Introducción a Python

Python es un lenguaje de programación de gran potencia, similar a Java, Ruby o Scheme. Aunque esta más orientado al objeto, es decir, está diseñado de tal manera que su sintaxis sea sencilla e intuitiva, permitiendo al programador centrarse en la resolución del problema y no en la forma del código [6].

Sus principales ventajas son:

- Se puede distribuir de manera libre, pudiendo leer sus códigos, realizarle cambios y mejoras, y utilizar parte de su código fuente en otros programas libres. Además, puede ser libremente modificado y redistribuido, de ahí su continuo progreso y aumento de prestaciones.
- Permite introducir módulos implementados en C o C++ en Python dentro de tu programa C o C++, lo que le aporta aún más facilidades a la hora de trabajar.
- Cuenta de manera estándar, con una extensa biblioteca con expresiones típicas, generación de documentos, evaluación de unidades, pruebas, procesos, bases de datos, navegadores web, correo electrónico, entre otras funciones dependientes del Sistema.

- Funciona en prácticamente todas las plataformas y sistemas operativos como Windows, Linux, MacOS y otros.
- Puede ser utilizado como lenguaje de programación en la interfaz de otros programas, como es el caso de ABAQUS.

Todas estas razones facilitan el uso de este lenguaje de programación y lo convierten en un lenguaje ideal para scripting y para el desarrollo rápido de aplicaciones.

4.3. Introducción a ABAQUS Scripting

ABAQUS Scripting es una variante dentro del programa de elementos finitos ABAQUS que permite crear los modelos introduciendo comandos del lenguaje de programación Python, en vez de trabajar con el módulo CAE, lo cual proporciona muchas facilidades y ventajas para modelos grandes.

ABAQUS Scripting puede usarse, o bien como complemento de Abaqus/CAE, utilizándose solo en los módulos que se crean necesarios, o bien como soporte único para crear un modelo.

La principal ventaja de trabajar con líneas de código en vez de con CAE, se encuentra a la hora de diseñar modelos de una determinada complejidad y que estarán sujetos a modificaciones a lo largo de su estudio. Esto incluye a la gran mayoría, ya que en un gran porcentaje de los trabajos realizados con programas de elementos finitos necesitan numerosos cambios, con el fin de alcanzar el modelo óptimo o las conclusiones requeridas en el estudio en cuestión. Por ello, la forma de funcionar de ABAQUS /CAE obliga a chequear, y a veces cambiar, todos los módulos del modelo cada vez que se introduce alguna modificación, lo cual supone un gran esfuerzo y, sobre todo, una pérdida de tiempo para simulaciones complicadas y largas como suelen ser con las que se trabaja en el mundo laboral.

Sin embargo, si se dispone de todo el trabajo en simples comandos organizados en líneas de código, modificar un modelo de simulación es algo bastante rápido y sencillo. Solo se ha de saber qué hay que modificar del

proyecto, acudir a la línea de código donde se ha introducido esa orden, y cambiar el viejo comando por el nuevo. De esta manera cuando se vuelva a ejecutar el trabajo, las modificaciones ya se habrán realizado.

```
#Variables
DimensionPlaca=0.1
GrosorPlaca=0.004
ProjectileHeight=0.025
ProjectileRadius=0.0065
```

Figura 4. 1: Variables de un código de Abaqus Scripting

En la Figura 4.1 puede observarse un ejemplo de las constantes de un modelo de Abaqus escrito mediante Python. Simplemente con variar el valor de alguna de ellas y de volver a compilar el archivo, el modelo se modificará inmediatamente.

Por otro lado, el lenguaje Python permite importar comandos guardados en otros archivos a modo de “Librería”. Esto supone poder organizar un trabajo muy grande en diferentes archivos más pequeños, de manera que sea mucho más cómodo trabajar con ellos. También así, es posible recurrir a comandos previamente utilizados para nuevos proyectos con geometrías, materiales o cargas similares.

Por último, un dato importante de los trabajos diseñados mediante ABAQUS Scripting es el tamaño que ocupan los archivos, ya que guardar un proyecto de cierta magnitud en formato CAE puede ocupar entorno a 350 MB. Sin embargo, un archivo de editor de texto, como NotePad++, donde guardar todos los comandos de un modelo no ocupa más de 50KB. Esta diferencia de tamaño para un mismo modelo, es una gran ventaja a la hora de compartir, transportar o trabajar.

Por estas razones, considero bastante importante saber utilizar ABAQUS Scripting como herramienta principal para diseñar modelos de simulación o como complemento para ABAQUS /CAE.

4.4. Desarrollo del Modelo

4.4.1. Introducción

A lo largo de este apartado se explicará el modelo de impacto que se ha realizado mediante ABAQUS, entrando a detallar cada uno de sus módulos, así como los comandos de Python utilizados.

El modelo simula las mismas situaciones de los ensayos experimentales, es decir, el impacto de un proyectil de forma cilíndrica contra la zona central de una placa cuadrada de la aleación de aluminio AA 5754-H111.Ésta se encuentra empotrada en sus cuatro lados y sus dimensiones están aquí definidas.

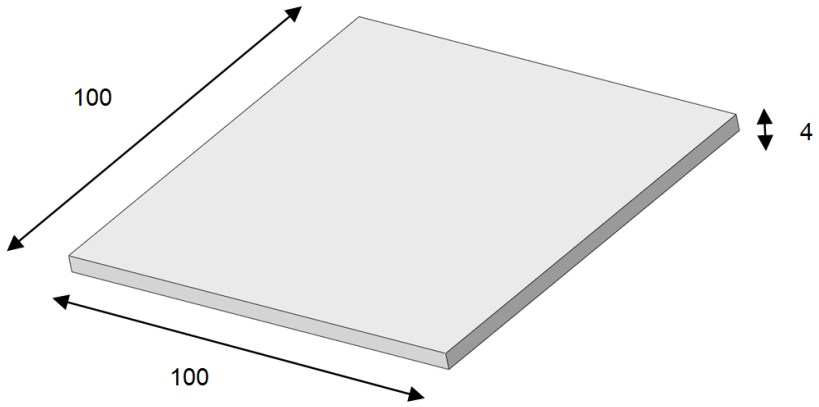


Figura 4. 2: Geometría y dimensiones de la placa en las simulaciones en mm

Los tres proyectiles utilizados tienen las dimensiones descritas en el ensayo experimental y una masa de 30g.

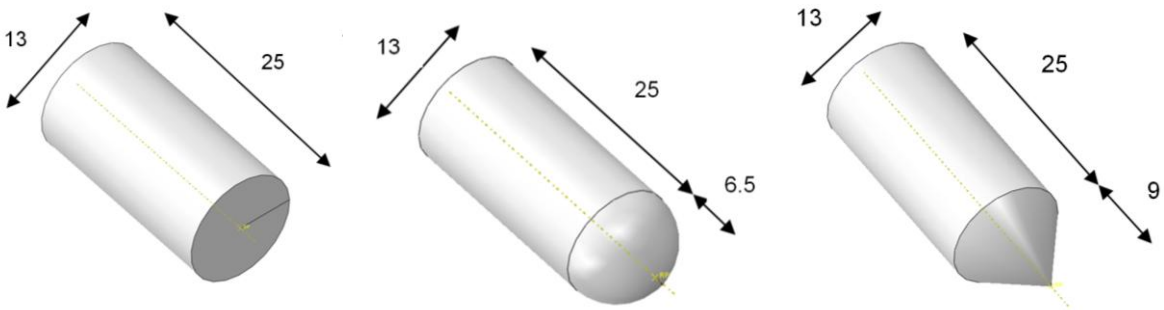


Figura 4. 3: Geometría y dimensiones de los proyectiles en las simulaciones en mm

El ángulo de incidencia del impacto es de 90º con la placa y su velocidad se ha ido variando durante el proyecto, tomando valores comprendidos entre 120 m/s y 200 m/s.

Las magnitudes que se quieren comparar en estas simulaciones son: la deformación de la placa, desplazamiento en la placa, energías en determinadas partes de la placa y velocidad de salida del proyectil.

A continuación se irá describiendo uno a uno los módulos que se usan en ABAQUS para diseñar un modelo y las líneas de código correspondientes al interfaz ABAQUS Scripting.

4.4.2. Part

En este módulo se definen las geometrías de los elementos que van a intervenir en la simulación. Cuando se trabaja con ABAQUS Scripting, la mejor manera de construir las partes es dibujar primero un boceto en dos dimensiones y después extruirlo o revolucionarlo, en función de la geometría a la que se quiera llegar.

Para la representación de los impactos, se han de crear los tres tipos de proyectiles y una única placa. En la placa se han realizado una serie de particiones y “Sets” que serán de utilidad para el posterior mallado y para el estudio de partes concretas de la placa. Se han definido tres particiones circulares, una del mismo diámetro del proyectil, otra un poco mayor y una última considerablemente mayor. Además también se ha divido la placa siguiendo sus dos diagonales para facilitar el mallado.

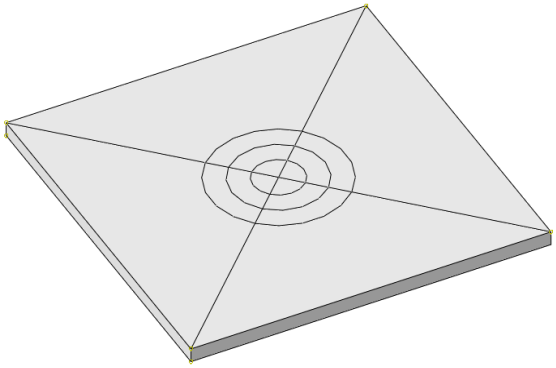


Figura 4. 4: Particiones de la placa en Abaqus

4.4.3. Property

En este punto del programa, hay que definir las propiedades de los materiales que se vaya a introducir en el modelo. Una vez hecho esto, se ha de asignar dichos materiales a las partes creadas en el módulo anterior.

Para el diseño del impacto, solo será necesario definir el material de la placa (AA 5754-H111), ya que el proyectil será definido como un sólido rígido, es decir, una pieza indeformable en el que no importan sus propiedades.

El criterio de endurecimiento que se ha utilizado para definir el comportamiento de la placa en las perforaciones ha sido la Ley de Endurecimiento de Johnson-Cook [7], la cual está siendo utilizada recientemente en numerosos estudios de simulación de análisis dinámico [8], [9].

Este modelo reproduce muy bien el endurecimiento de metales cuando existen grandes velocidades y deformaciones. La formulación de este método se expresa por las siguientes ecuaciones.

$$\bar{\sigma}\left(\bar{\varepsilon}^{p}, \dot{\bar{\varepsilon}}^{p}, \Theta\right) = \left[A + B(\bar{\varepsilon}^{p})^{n}\right]\left[1 + C\ln\left(\frac{\dot{\bar{\varepsilon}}^{p}}{\dot{\varepsilon}_{0}}\right)\right]\left(1 - \Theta^{m}\right) \qquad (1)$$

$$\Theta = \frac{T - T_{0}}{T_{m} - T_{0}} \qquad (2)$$

Donde A es el límite elástico en condiciones cuasiestáticas, B es el módulo de endurecimiento por deformación, n es un coeficiente de endurecimiento, C indica la sensibilidad a la velocidad de formación y m la sensibilidad a la temperatura.

Como puede observarse, la ecuación consta de tres términos. El primero representa la importancia de la deformación en el endurecimiento, el segundo la sensibilidad de la deformación y el último la sensibilidad a la temperatura.

Las propiedades utilizadas para la aleación AA5757-H111 son las siguientes:

Material	A (MPa)	B (MPa)	n (-)	C (-)	ε_0 (s^{-1})	T_0 (K)	T_m (K)	m (-)
5754-H111	28.13	278.67	0.183	0.00439	0.1	293	873	2.527

β (-)	C_p ($Jkg^{-1}K^{-1}$)	ρ (kgm^{-3})
0.9	900	2700

Tabla 4. 1: La tabla muestra en la parte superior las constantes de Johnson Cook específicas de esta aleación. Abajo, las propiedades comunes para todos los aluminios [10].

4.4.4. Assembly

Aquí se sitúan todas las piezas creadas de forma que reproduzcan el caso real que se quiera simular de la manera más veraz posible.

Para este caso concreto de estudio, se ha colocado el proyectil en el centro de la placa y con una separación vertical muy pequeña respecto a esta (0.5 mm).

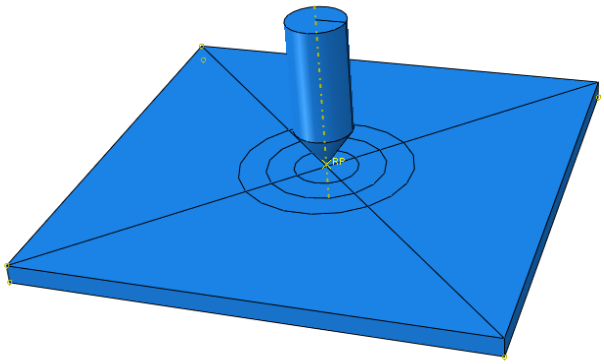


Figura 4. 5: Montaje de la placa y el proyectil cónico

4.4.5. Step

En el módulo “Step” se crearán los “pasos” o “módulos de tiempo” en los que se aplicarán las cargas y las condiciones de contorno. Esto dependerá del tiempo de acción de las mismas.

Para este análisis se han creado dos “Steps”: “Initial” e “Impact” dependiendo de si la carga o condición de contorno se aplica durante toda la simulación (“Initial”) o solamente durante el periodo de tiempo en el que se produce el impacto (“Impact”).

4.4.6. Interaction

En esta parte debemos definir la forma de interaccionar que hay entre las distintas partes creadas del modelo.

Para los ensayos de perforación, hay que definirlo como “Hard Contact”, es decir, que el programa detecte un choque entre ambas partes cuando éstas entren en contacto.

4.4.7. Load and Boundary Condition

En este apartado debemos definir todas las condiciones de contorno, cargas o movimientos lineales y angulares a los que se encuentra sometido el modelo. Indicando magnitud, dirección, sentido y punto o puntos de aplicación.

Como ya se ha señalado anteriormente, la placa está sujeta y se le impide cualquier movimiento, por lo que en el programa se ha de restringir todos los grados de libertad de la placa ("Encastre"). Por otro lado, el proyectil también tendrá restringido todos los grados de libertad, a excepción del vertical, donde se le impondrá una velocidad que durante las simulaciones irá variando.

4.4.8. Mesh

En este módulo del programa, la definición y modelización del problema como tal ya ha sido acabada y ahora, se debe definir el mallado, es decir, elegir el número de elementos finitos en los que queremos dividir nuestro modelo. Dependiendo del número de nodos que se elijan, el programa tardará más o menos en calcular los resultados, por lo que para ser lo más eficiente posible, se ha de realizar un mallado más fino en las zonas del modelo que sean críticas y reducir el número de nodos en las zonas de menor interés.

Para todos los ensayos de ABAQUS, se ha utilizado el mismo mallado. Por tema de eficiencia se ha hecho un mallado por partes, variando el número de nodos y el tipo del elemento en función de la proximidad a la zona de impacto. En la zona central, con un diámetro similar al del proyectil se ha elegido un tipo de elemento C3D4, ya que no se quiere predefinir la dirección del fallo. En la zona inmediatamente contigua a la del impacto, se ha utilizado un tipo de elemento octaédrico tipo C3D8R. Para el resto de la placa, que coincide con las partes más alejadas, se ha asignado también un elemento octaédrico de tipo C3D8R, pero con unas dimensiones mayores, ya que se requieren menos nodos.

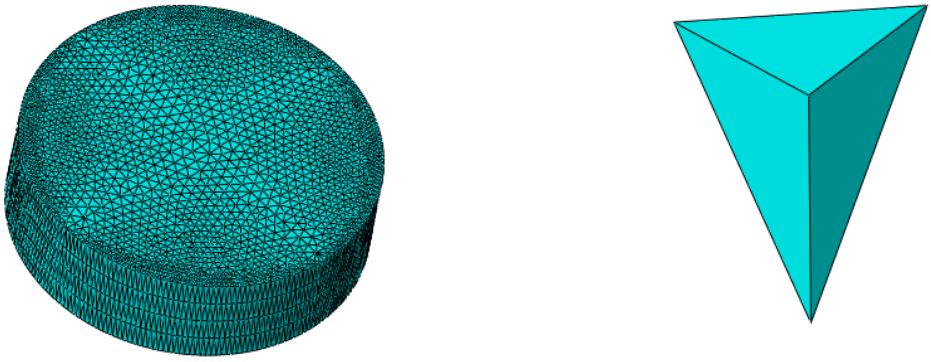


Figura 4. 6: Zona de la placa del mismo diámetro que el proyectil y el elemento utilizado C3D4

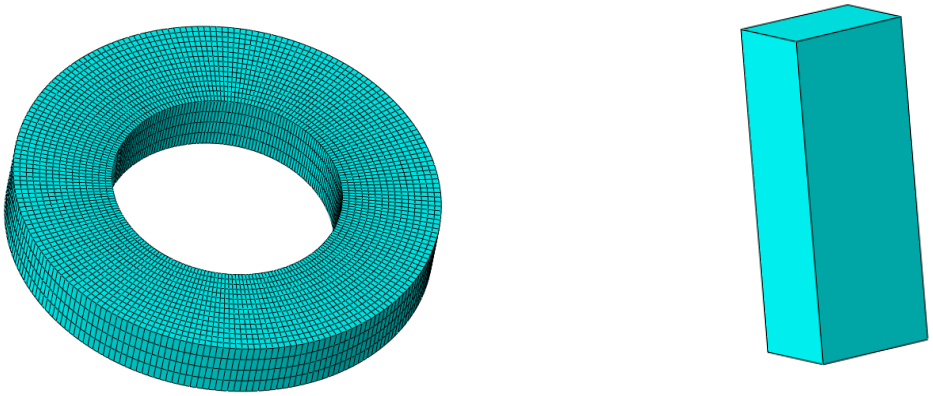


Figura 4. 7: Zona de la placa próxima al impacto y el elemento utilizado C3D8R

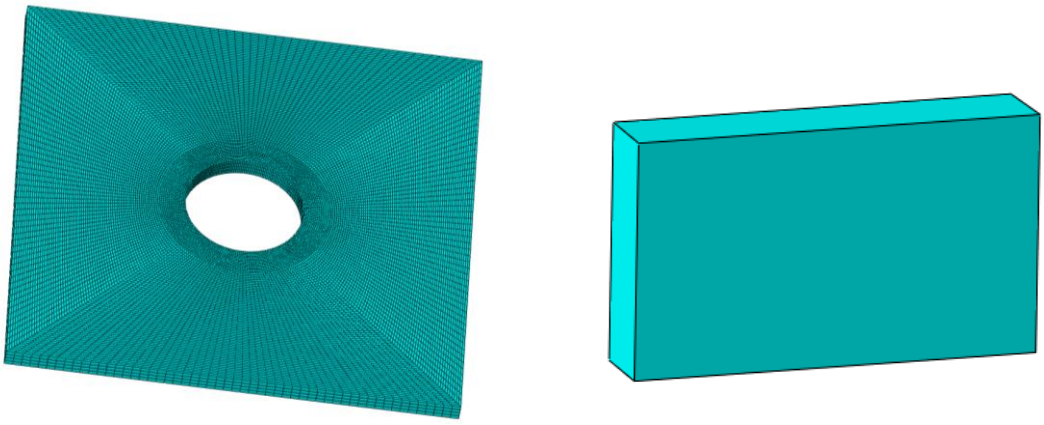
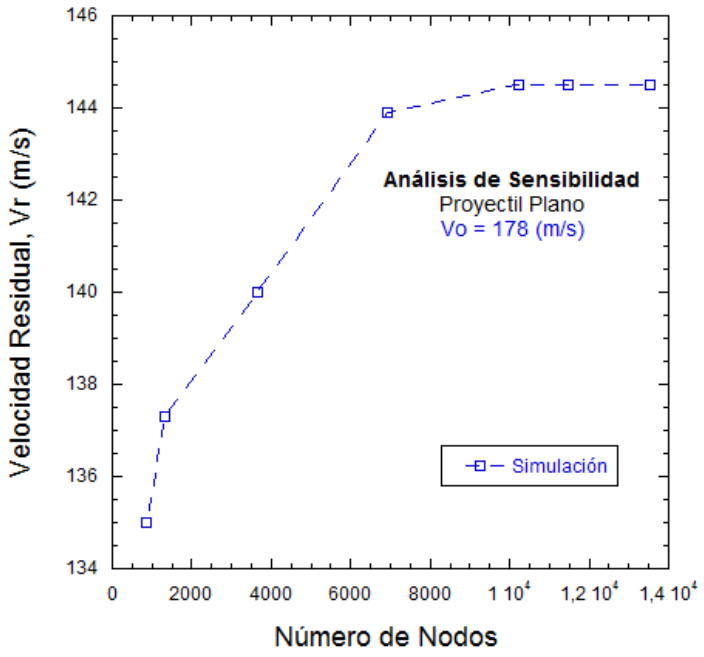


Figura 4. 8: Resto de la placa y el elemento utilizado C3D8R

Por otro lado, se ha realizado un análisis de sensibilidad del mallado, para obtener una malla que garantizara resultados reales, pero teniendo el menor número de nodos posibles y así reducir los tiempos de cálculo. Este análisis se ha realizado manteniendo siempre el proyectil plano y con una misma velocidad inicial, y se ha ido aumentando el número de nodos en la zona del impacto, hasta que los valores obtenidos de velocidad residual se han estabilizado, sabiéndose así que se ha alcanzado una malla fiable.



Gráfica 4. 1

Como puede observarse en la gráfica, la velocidad residual se estabiliza a partir de los 10000 nodos en la zona central de la placa, en un valor de velocidad residual de 144,5 m/s. Para los ensayos, basándonos en los resultados de la gráfica, se ha optado por utilizar el número de nodos del punto intermedio de los tres que tienen velocidad residual constante. Para este punto hay un total de 11471 nodos en la zona del impacto, un total de 17600 en la parte inmediatamente contigua a éste y 93500 en la parte periférica de la placa.

4.4.9. Job

Este apartado del programa está reservado para poder guardar el trabajo realizado y poder "lanzarlo" para comenzar los cálculos.

4.4.10. Visualization

Este módulo de ABAQUS es el único que es común para ABAQUS /CAE y para ABAQUS Scripting, ya que no se utiliza para diseñar ni definir ninguna parte del trabajo, sino para observar los resultados obtenidos por el programa.

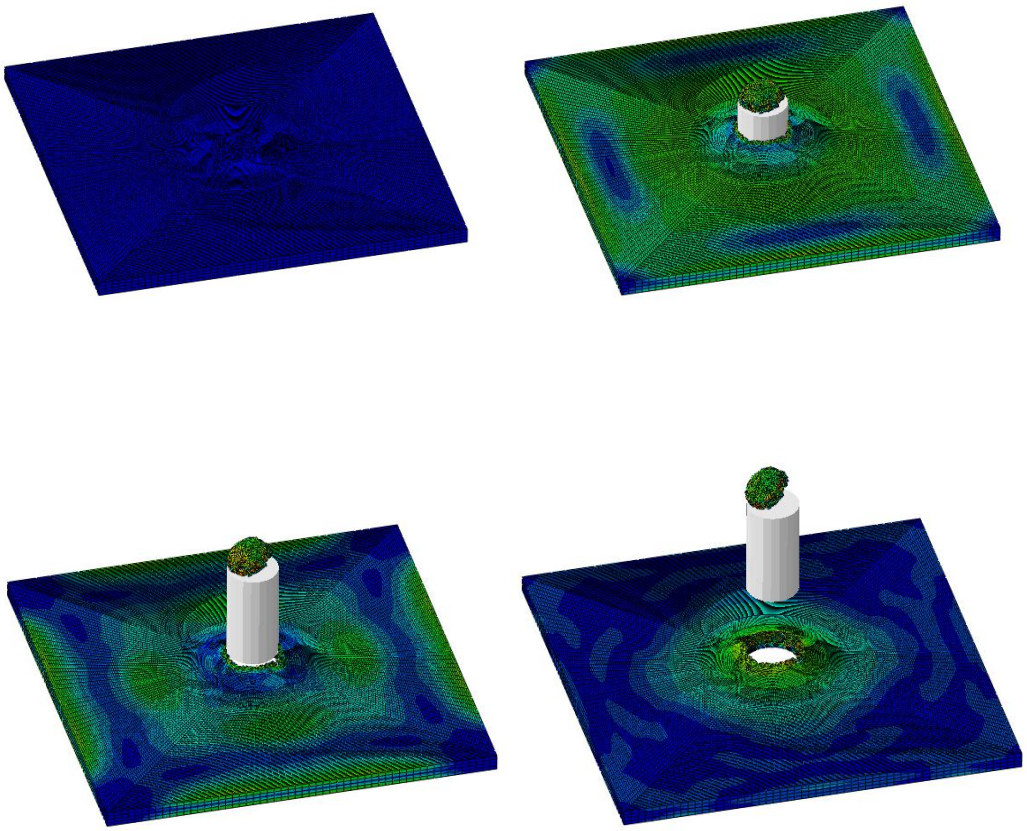


Figura 4. 9: Evolución de la perforación de la placa vista en el modulo "Visualization"

4.5. Descripción del modelo

4.5.1. Introducción

En este epígrafe se explicará el código utilizado paso por paso adjuntando el fragmento correspondiente [11], [12]. Se ha dividido en los módulos de ABAQUS que se han nombrado anteriormente y algún apartado más que ha sido necesario incluir. El script va siendo ejecutado de arriba abajo por el programa y aquellas líneas que tienen en su inicio el símbolo "#" no son compiladas, de manera que sirven para indicar en qué parte del modelo estamos y alguna aclaración si fuera necesaria. A parte de eso, se podrá observar que al inicio de cada módulo hay una sentencia que comienza con el comando "import" y seguida del nombre del módulo en el que estamos trabajando, esto sirve para que el programa importe la librería correspondiente a cada módulo de trabajo.

El modelo aquí descrito, es tal y como se ha explicado en el punto anterior, añadiendo que los tres tipos de ensayo están incluidos en un único código, pudiéndose elegir la geometría del proyectil y otras características del modelo mediante preguntas que nos realizará el programa según vaya ejecutando el código.

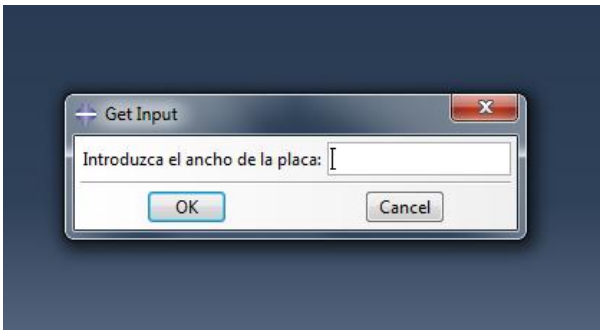


Figura 4. 10: Preguntas realizadas por el programa para definir los parámetros del modelo

4.5.2. Inicialización

El tramo de código encargado de esta función es el siguiente.

```
# Placa de aluminio de impactos

from abaqus import *
from abaqusConstants import *
import regionToolset
import math

backwardCompatibility.setValues(includeDeprecated=True,
reportDeprecated=False)

session.viewports['Viewport:1'].setValues(displayedObject=None)
```

Estas sentencias sirven para que el programa ABAQUS reconozca el código. Las cinco primeras permiten importar la librería por defecto de los comandos de ABAQUS y también sus constantes. Además, sirve para llamar a las funciones y constantes básicas matemáticas que será necesario utilizar en ciertas partes del código para describir geometrías.

Por otro lado, la última línea sirve para activar la ventana de visualización en el ABAQUS /CAE. De esta manera, si se quiere ir comprobando nuestro modelo o ver los errores que han podido aparecer, se podrá abrir el archivo en ABAQUS /CAE y visualizar el código ya compilado.

4.5.3. Crear el modelo

El siguiente bloque crea el modelo

```
#Create model

mdb.models.changeKey(fromName='Model-1', toName='Impact')
shellModel = mdb.models['Impact']
```

Como se puede ver, en estas dos órdenes, se crea un modelo y se le cambia el nombre, pasando a ser “Impact” del que viene por defecto “Model-1”.

4.5.4. Elección de los parámetros y creación del proyectil

Esta parte del código sirve para determinar parámetros importantes y que pueden variar en función del ensayo: la geometría de la placa, velocidad y masa del proyectil, y cuál de los tres proyectiles se querrá utilizar. Todo ello mediante una serie de preguntas que nos realizará el programa. En función de las respuestas, se asignará el valor de las variables que se van a usar durante el modelo. Las líneas de código que están precedidas del símbolo “#”, sirven para mostrar los valores con los que se ha trabajado en estas simulaciones.

```
#Answers

DimensionPlaca=float(getInput('Introduzca el ancho de la placa:'))
#DimensionPlaca=0.1
GrosorPlaca=float(getInput('Introduzca el grosor de la placa:'))
#GrosorPlaca=0.004
ProjectileMass=float(getInput('Introduzca la masa del proyectil:'))
#ProjectileMass=0.03
ProjectileVelocity=float(getInput('Introduzca la velocidad del proyectil:'))
ProjectileVelocity= (-ProjectileVelocity)
#ProjectileVelocity=-179.86
ProjectileShape=float(getInput('Introduzca el la forma del proyectil.
Blunt=1, Conical=2, Hemispherical=3:'))
#ProjectileShape='x'

inercia1=1e-9
inercia2=1e-9
inercia3=1e-9
```

Estas órdenes indican al programa que pregunte la frase que está entrecomillada en cada línea. La respuesta introducida por el usuario será guardada en la variable a la que está igualada el comando.

En las primeras sentencias, se preguntan las dimensiones de la placa, la masa del proyectil y la velocidad del proyectil. La última sirve para elegir el tipo de geometría del proyectil y asigna la respuesta que se dé (que será 1,

2 o 3) a la variable “x”. Como se verá en los tramos de código siguientes, esta variable entrará dentro de una función “if”, eligiendo un camino u otro en función del valor que se le haya dado. Las últimas sentencias son los valores de la inercia, que son los mismos para todos los ensayos, pero que el programa los pide para poder realizar los cálculos.

Esta es la parte del código correspondiente a elegir el proyectil plano.

```
if ProjectileShape == 1 :
 #Variables

 ProjectileHeight=float(getInput('Introduzca la altura del proyectil:'))
 #ProjectileHeight=0.025
 ProjectileRadius=float(getInput('Introduzca el radio del proyectil:'))
 #ProjectileRadius=0.0065
 Time= (0.0005+GrosorPlaca+ProjectileHeight+ProjectileHeight)/(-
ProjectileVelocity)

 #-------------------------------------------------------

 #Create projectile

 import sketch
 import part

 projectileProfileSketch =
 shellModel.ConstrainedSketch(name='projectileProfileSketch',sheetSize=5.0)
 projectileProfileSketch.ConstructionLine(point1=(0.0,-100.0),
 point2=(0.0,100.0))
 projectileProfileSketch.Line(point1=(0,0), point2=(ProjectileRadius,0))
 projectileProfileSketch.Line(point1=(ProjectileRadius,0),
 point2=(ProjectileRadius,ProjectileHeight))
 projectileProfileSketch.Line(point1=(ProjectileRadius,ProjectileHeight),
 point2=(0,ProjectileHeight))

 projectilePart= shellModel.Part(name='Projectile', dimensionality=THREE_D,
 type=ANALYTIC_RIGID_SURFACE)
 projectilePart.AnalyticRigidSurfRevolve(sketch=projectileProfileSketch)
```

Si se ha elegido “x=1”, el programa compilará los comandos que aquí se muestran. En la primera parte, el programa continúa haciendo preguntas sobre las dimensiones del proyectil.

El comando “Time”, representa el tiempo de cálculo del programa, y se ha automatizado en función de la velocidad inicial y del espacio que tiene que recorrer el proyectil con un margen de seguridad, para evitar que el tiempo de cálculo sea inferior al deseado.

En cambio en la parte encabezada por la sentencia “#Create projectile” se crea dicho proyectil, utilizando las variables mencionadas. Para la creación del mismo, primero se guarda un boceto dibujado mediante órdenes, después éste se revoluciona en torno a un eje también definido.

La manera de proceder con las otras dos geometrías es idéntica, con la única diferencia de que algunas preguntas y variables varían. Aquí se adjuntan ambos códigos respectivamente.

```
if ProjectileShape == 2 :
 #Variables

 ProjectileTotalHeight=float(getInput('Introduzca la altura total del
proyectil:'))
 #ProjectileTotalHeight=0.034
 ProjectileRadius=float(getInput('Introduzca el el radio del proyectil:'))
 #ProjectileRadius=0.0065
 ProjectileAngle=float(getInput('Introduzca el angulo de punta del
proyectil:'))
 #ProjectileAngle=36
 ProjectileHeight1=ProjectileRadius/(tan(ProjectileAngle*(math.pi)/180))
 ProjectileHeight=(ProjectileTotalHeight-ProjectileHeight1)
 Time= (0.0005+GrosorPlaca+ProjectileTotalHeight+ProjectileTotalHeight)/(-
ProjectileVelocity)

 #-------------------------------------------------------------------------

 #Create projectile

 import sketch
 import part

 projectileProfileSketch =
shellModel.ConstrainedSketch(name='projectileProfileSketch',sheetSize=5.0)
 projectileProfileSketch.ConstructionLine(point1=(0.0,-100.0),
point2=(0.0,100.0))
 projectileProfileSketch.Line(point1=(0,0),
point2=(ProjectileRadius,ProjectileHeight1))
 projectileProfileSketch.Line(point1=(ProjectileRadius,ProjectileHeight1),
point2=(ProjectileRadius,ProjectileHeight+ProjectileHeight1))

projectileProfileSketch.Line(point1=(ProjectileRadius,ProjectileHeight+Projec
tileHeight1), point2=(0,ProjectileHeight+ProjectileHeight1))

 projectilePart= shellModel.Part(name='Projectile', dimensionality=THREE_D,
type=ANALYTIC_RIGID_SURFACE)
 projectilePart.AnalyticRigidSurfRevolve(sketch=projectileProfileSketch)
```

```
if ProjectileShape == 3 :
 #Variables

 ProjectileTotalHeight=float(getInput('Introduzca la altura total del
proyectil:'))
 #ProjectileTotalHeight=0.025+ProjectileRadius
 ProjectileRadius=float(getInput('Introduzca el radio del proyectil:'))
 #ProjectileRadius=0.0065
 ProjectileHeight=(ProjectileTotalHeight-ProjectileRadius)
 Time= (0.0005+GrosorPlaca+ProjectileHeight+ProjectileHeight)/(-
ProjectileVelocity)

#-------------------------------------------------------------------

#Create projectile

 import sketch
 import part

 projectileProfileSketch =
shellModel.ConstrainedSketch(name='projectileProfileSketch',sheetSize=5.0)
 projectileProfileSketch.ConstructionLine(point1=(0.0,-100.0),
point2=(0.0,100.0))
 projectileProfileSketch.ArcByCenterEnds(center=(0,ProjectileRadius),
point1=(0,0), point2=(ProjectileRadius,ProjectileRadius))
 projectileProfileSketch.Line(point1=(ProjectileRadius,ProjectileRadius),
point2=(ProjectileRadius,ProjectileHeight+ProjectileRadius))

projectileProfileSketch.Line(point1=(ProjectileRadius,ProjectileHeight+Projec
tileRadius), point2=(0,ProjectileHeight+ProjectileRadius))

 projectilePart= shellModel.Part(name='Projectile', dimensionality=THREE_D,
type=ANALYTIC_RIGID_SURFACE)
 projectilePart.AnalyticRigidSurfRevolve(sketch=projectileProfileSketch)
```

4.5.5. Creación de la placa y particiones

Utilizando las variables definidas anteriormente, se define el boceto de un rectángulo, en este caso cuadrado, que posteriormente es extruído con el grosor que se ha elegido. Aquí se muestra el código correspondiente a estas acciones.

```
#Create the shell

import sketch
import part

shellProfileSketch = shellModel.ConstrainedSketch(name='Shell CS
Profile',sheetSize=5)
shellProfileSketch.rectangle(point1=(0,0),
point2=(DimensionPlaca,DimensionPlaca))
shellPart=shellModel.Part(name='Placa',dimensionality=THREE_D,
type=DEFORMABLE_BODY)
shellPart.BaseSolidExtrude(sketch=shellProfileSketch, depth=GrosorPlaca)
```

Como se ha explicado en el desarrollo del modelo, para el módulo “Interaction”, el mallado y para requerir solicitaciones en algunas partes concretas de la placa, es necesario realizar particiones en ésta y definir “Sets”. La geometría de las mismas es la que se ha explicado anteriormente y los comandos corresponden a los especificados por ABAQUS Scripting. Primero se han definido las circulares y se han definido los “Sets”, y posteriormente se han hecho las particiones diagonales. Esta parta del código está adjuntado al final en el Anexo A.

4.5.6. Definición y asignación del material

Para definir el material, solo hay que nombrarlo e ir indicando sus propiedades tal y como se definió en el epígrafe anterior. Aquí se adjunta el tramo de código correspondiente.

```
#Create shell material material

import material
import odbMaterial

shellMaterial=shellModel.Material(name='AA 5754-H111')
shellMaterial.Density(table=((2700, ), ))
shellMaterial.Elastic(table=((7e+10,0.3), ))
shellMaterial.InelasticHeatFraction(fraction=0.9)
shellMaterial.Plastic(hardening=JOHNSON_COOK, table=((2.813E+07, 278.67E+06,
0.183, 2.527, 873, 293), ))
shellMaterial.plastic.RateDependent(table=((0.00439, 0.1), ),
type=JOHNSON_COOK)
shellMaterial.SpecificHeat(table=((900, ), ))
```

Después, se debe asignar dicho material a la placa ya que, como se ha explicado anteriormente, el proyectil funciona como un sólido indeformable. Para relacionar la aleación AA 5754-H111 con la placa, se ha de crear una sección y asignándole el material creado e indicar la geometría a la que se desea unir. Aquí se adjunta dicho proceso.

```
#Create the section

import section

shellSection = shellModel.HomogeneousSolidSection(name='Shell Section',
material='AA 5754-H111')
shell_region = (shellPart.cells,)
shellPart.SectionAssignment(region=shell_region, sectionName='Shell Section')
```

4.5.7. Ensamble del modelo

Este tramo de líneas de código representa el ensamblado de la placa y el proyectil, situándolos del mismo modo que en los ensayos experimentales.

```
#Create assembly

import assembly

shellAssembly = shellModel.rootAssembly
shellInstance = shellAssembly.Instance(name='Shell Instance', part=shellPart,
dependent=ON)

projectileInstance = shellAssembly.Instance(name='Projectile Instance',
part=projectilePart, dependent=ON)
shellAssembly. rotate(instanceList=('Projectile Instance',), axisPoint=
(0,0,0), axisDirection= (0.001,0,0), angle=(90))
shellAssembly. translate(instanceList=('Projectile Instance',),
vector=(DimensionPlaca/2,DimensionPlaca/2,GrosorPlaca+0.0005))
```

Como se puede observar, se crea un ensamblado en el modelo y se introducen las partes, ambas situadas en el eje de coordenadas. A partir de ahí, se introducen sentencias que obliguen a las partes, mediante vectores, a rotar o desplazarse hasta alcanzar la disposición deseada.

4.5.8. Creación del “Step” y de la interacción

En la sentencia de código facilitada a continuación, se crea el “Step” correspondiente al impacto (definido como “Impact”), ya que el “Step” inicial se crea por defecto al iniciar un modelo. La interacción es la que se ha definido anteriormente, y tendrá lugar entre todas las superficies del proyectil y todas las superficies que se encuentren dentro de la partición circular intermedia (definido como “Set-3”).

```
#Create the step

import step

shellModel.ExplicitDynamicsStep(name='Impact', previous='Initial',
timePeriod=Time)

#-------------------------------------------------------------------------

#Create interaction

shellModel.ContactProperty('IntProp-1')
shellModel.interactionProperties['IntProp-1'].TangentialBehavior(
    formulation=PENALTY, directionality=ISOTROPIC, slipRateDependency=OFF,
    pressureDependency=OFF, temperatureDependency=OFF, dependencies=0,
table=((
    0.1, ), ), shearStressLimit=None, maximumElasticSlip=FRACTION,
    fraction=0.005, elasticSlipStiffness=None)
shellModel.interactionProperties['IntProp-1'].NormalBehavior(
    pressureOverclosure=HARD, allowSeparation=ON,
    constraintEnforcementMethod=DEFAULT)

side1Faces1 = projectileInstance.faces
region1=regionToolset.Region(side1Faces=side1Faces1)
region2=shellInstance.sets['Set-3']
shellModel.SurfaceToSurfaceContactExp(name ='Int-1',
    createStepName='Impact', master = region1, slave = region2,
    mechanicalConstraint=PENALTY, sliding=FINITE,
    interactionProperty='IntProp-1', weightingFactorType=SPECIFIED,
    weightingFactor=1, initialClearance=OMIT, datumAxis=None,
    clearanceRegion=None)
```

4.5.9. Solicitaciones del modelo

En estos impactos, se están estudiando una serie de magnitudes en la placa y proyectil tras el choque. Para que el programa nos calcule los valores de estas magnitudes, tienen que ser definidas con anterioridad.

Este primer tramo de código, solicita al programa que calcule unas determinadas magnitudes para todos los nodos del modelo.

```
#Create the field output request

shellModel.fieldOutputRequests.changeKey(fromName='F-Output-1',
toName='Selected Field Outputs')
shellModel.FieldOutputRequest(name='Selected Field Outputs',
createStepName='Impact', numIntervals=5)
shellModel.fieldOutputRequests['Selected Field
Outputs'].setValues(variables=('PE', 'PEEQ', 'S', 'STATUS', 'TRIAX', 'U',
'V'))
```

Símbolos	Magnitud
PE	Deformación plástica
PEEQ	Deformación plástica equivalente
S	Todas las componentes de la tensión
STATUS	Estado del elemento (Si es activo o no)
TRIAX	Triaxialidad
U	Todas las componentes del desplazamiento
V	Todas las componentes de la Velocidad

Tabla 4. 2: Tabla de equivalencias de las magnitudes solicitadas a todo el modelo [13]

El segundo bloque sirve para que el programa nos facilite las energías absorbidas en partes concretas de la placa. Estas partes son, la partición circular de mismo diámetro que el proyectil (“Set-1”) y el resto de la placa (“Set-2”).

```
#Create the history output request

region1 = shellInstance.sets['Set-1']
region2 = shellInstance.sets['Set-2']
shellModel.HistoryOutputRequest(name='H-Local', createStepName='Impact',
variables=('ALLIE','ALLKE',), region=region1, sectionPoints=DEFAULT,
rebar=EXCLUDE )
shellModel.HistoryOutputRequest(name='H-Global', createStepName='Impact',
variables=('ALLIE','ALLKE',), region=region2, sectionPoints=DEFAULT,
rebar=EXCLUDE )
```

Símbolos	Magnitud
ALLIE	Energía total de deformación
ALLKE	Energía Cinética

Tabla 4. 3: Tabla de equivalencias de las magnitudes solicitadas a las regiones de la placa [13]

4.5.10. Definición de las condiciones de contorno

En este tramo de código se define el empotramiento de la placa en sus cuatro superficies laterales y la velocidad e inercia del proyectil. Aquí se adjunta el tramo correspondiente a la placa.

```
#Create Encaster

fixed_side_face1 = shellInstance.faces.findAt(((0, DimensionPlaca/2,
GrosorPlaca/2), ))
fixed_end_face_region=regionToolset.Region(faces=fixed_side_face1)
shellModel.EncastreBC(name='Encaster one', createStepName='Impact',
region=fixed_end_face_region)

fixed_side_face2 = shellInstance.faces.findAt(((DimensionPlaca/2, 0,
GrosorPlaca/2), ))
fixed_end_face_region=regionToolset.Region(faces=fixed_side_face2)
shellModel.EncastreBC(name='Encaster two', createStepName='Impact',
region=fixed_end_face_region)

fixed_side_face3 = shellInstance.faces.findAt(((DimensionPlaca/2,
DimensionPlaca, GrosorPlaca/2), ))
fixed_end_face_region=regionToolset.Region(faces=fixed_side_face3)
shellModel.EncastreBC(name='Encaster three', createStepName='Impact',
region=fixed_end_face_region)

fixed_side_face4 = shellInstance.faces.findAt(((DimensionPlaca,
DimensionPlaca/2, GrosorPlaca/2), ))
fixed_end_face_region=regionToolset.Region(faces=fixed_side_face4)
shellModel.EncastreBC(name='Encaster four', createStepName='Impact',
region=fixed_end_face_region)
```

Como se puede observar, para la sujeción de la placa se han realizado cuatro encastres, uno en cada superficie lateral que ha sido designada mediante un punto que contenga dicha cara, gracias al comando “shellInstance.faces.findAt”.

Sin embargo, para definir el desplazamiento del proyectil, primero se ha de definir un punto de referencia donde se puedan asignar estas condiciones de contorno. Una vez hecho esto se restringirán el movimiento en todas direcciones menos en el eje Y, donde se le asigna la variable velocidad determinada en las preguntas iniciales.

```
#Projectile Displacement

r = shellModel.rootAssembly.instances['Projectile Instance'].referencePoints
refPoints=(r[2], )
regionRF = regionToolset.Region(referencePoints=refPoints)

shellModel.DisplacementBC(name='Projectile displacement',
createStepName='Impact', region=regionRF, u1=0, u2=0, ur1=0, ur2=0, ur3=0)

#Predefined Field, Velocity

shellModel.Velocity(name='Velocity', region=regionRF, velocity1=0,
velocity2=0, velocity3=ProjectileVelocity, omega=0, axisBegin=(0,0,0),
axisEnd=(0,0,0))
```

La sentencia de la inercia consiste simplemente en introducir los valores definidos al principio para toda la geometría del proyectil.

```
#Projectile Mass

projectilePart.engineeringFeatures.PointMassInertia(name='Inertia-1',
region=regionRF, mass=ProjectileMass, i11=inercia1, i22=inercia2,
i33=inercia3, alpha=0.0, composite=0.0)
```

4.5.11. Creación del mallado

El mallado es una de las partes del código más tediosas, ya que se ha tenido que ir seleccionando cada eje por separado para poder asignarle el número de nodos correspondientes. Además, hay que tener en cuenta que la placa se ha mallado con tres tipos de elementos diferentes en función de su posición. Por ello, como se podrá ver más adelante en el código, los ejes se han agrupado de cuatro en cuatro, según su posición y su número de nodos.

Las primeras sentencias sirven para definir los tipos de nodos que se usarán y también para determinar que el mallado se realizará en toda la placa.

```
#Create the mesh

import mesh

elemType1 =mesh.ElemType(elemCode=C3D8R, elemLibrary=EXPLICIT,
kinematicSplit=AVERAGE_STRAIN, secondOrderAccuracy=OFF,
hourglassControl=DEFAULT, distortionControl=DEFAULT)
elemType2 =mesh.ElemType(elemCode=C3D4, elemLibrary=EXPLICIT,
secondOrderAccuracy=OFF,  distortionControl=DEFAULT)
selectedShellCells=shellPart.cells.findAt((0, 0, 0),)
shellMeshRegion=(selectedShellCells,)
```

Las seis siguientes son para definir las regiones en las que se usará cada tipo de elemento (para lo que se han utilizado los “Sets”).

```
h=Set3.cells
p=Set5.cells

shellPart.setElementType(regions=Set2, elemTypes=(elemType1,))
shellPart.setElementType(regions=Set1, elemTypes=(elemType2,))

shellPart.setMeshControls(regions=h, elemShape=TET, technique=FREE)
shellPart.setMeshControls(regions=p, elemShape=HEX_DOMINATED,
technique=STRUCTURED)
```

El resto del código, a excepción de la última línea, define el número de divisiones por eje agrupándolos de cuatro en cuatro como se ha explicado antes. Estas sentencias son adjuntadas en Anexo B.

El último comando se encarga de generar la malla que hemos definido con todas las sentencias anteriores.

```
shellPart.generateMesh()
```

4.5.12. Creación del Job

Es la parte final del código, y sirve para crear un archivo desde el que iniciar los cálculos. En este caso, el nombre del archivo es “ShellImpactJob”. También se puede programar para que en cuanto abras el archivo en ABAQUS /CAE, empiece a realizar los cálculos, pero se ha preferido no poner, por si hace falta hacer alguna modificación final.

```
#Create and run the job

import job

mdb.Job(name='ShellImpactJob', model='Impact', type=ANALYSIS,
explicitPrecision=SINGLE, nodalOutputPrecision=SINGLE, description='Job
simulates a proyectile impact', parallelizationMethodExplicit=DOMAIN,
multiprocessingMode=DEFAULT, numDomains=1, userSubroutine='', numCpus=1,
memory=50, memoryUnits=PERCENTAGE, scratch='', echoPrint=OFF, modelPrint=OFF,
contactPrint=OFF, historyPrint=OFF)

mdb.jobs['ShellImpactJob'].writeInput(consistencyChecking=OFF)
```

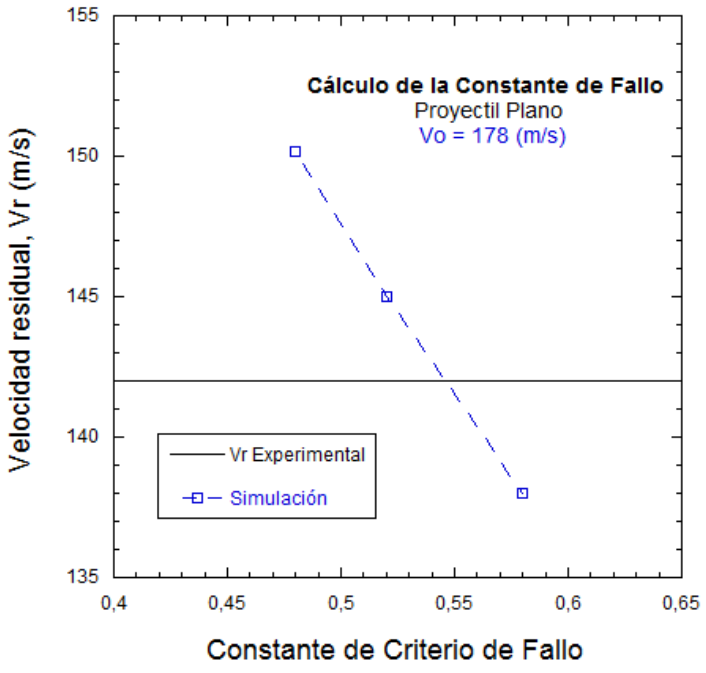
4.6. Constante de Fallo

Una vez se ha finalizado el diseño del modelo, se realizará un analisis de la deformación de fallo. Para ello se realiza una iteración de éste valor de manera que los valores numéricos se ajusten con el mínimo error a los resultados experimentales. Existen numerosos criterios de fallo para intentar reproducir el modo de fallo de un material en los modelos de simulación de ABAQUS. En función de las necesidades de cada cual, se deberá hacer uso de uno u otro. Para los cálculos numéricos de estos impactos, se ha elegido el modelo de fallo dinámico de Johnson-Cook, que funciona muy bien en la mayoría de situaciones dinámicas y concretamente en altas velocidades de deformación en los metales.

El criterio de fallo del material viene representado matemáticamente por la constante ε, cuyo valor depende principalmente del material y del estado tensional, en este caso de la placa.

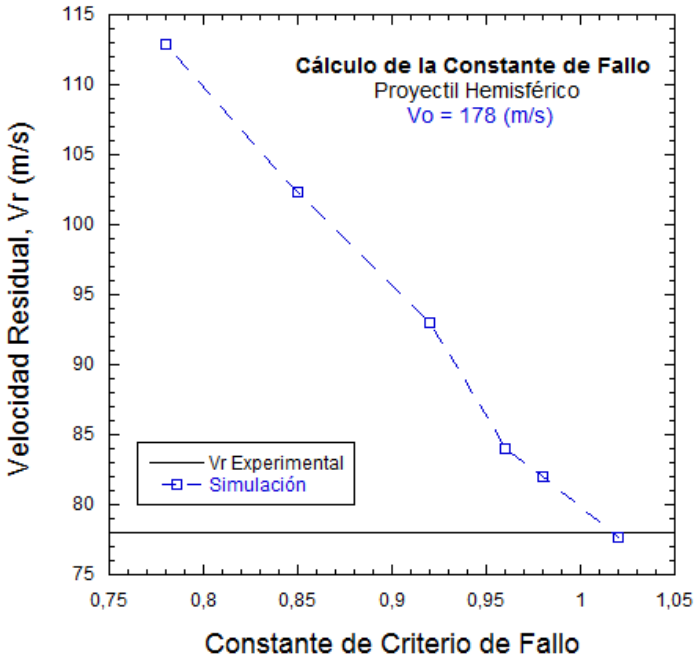
Esto implica que aunque los ensayos se han realizado siempre sobre una placa de una misma aleación de aluminio, la constante Ɛ del criterio de fallo variará dependiendo de la geometría del proyectil, ya que estos provocan tensiones distintas sobre la placa.

El cálculo de la constante del criterio de fallo correspondiente a cada proyectil, se ha determinado de una manera empírica. Se ha escogido una velocidad inicial para cada una de las geometrías, y se ha ido variando el valor de Ɛ, hasta que la velocidad residual de dicho proyectil ha coincidido con la de los ensayos experimentales, tal y como se muestra en las siguientes gráficas.

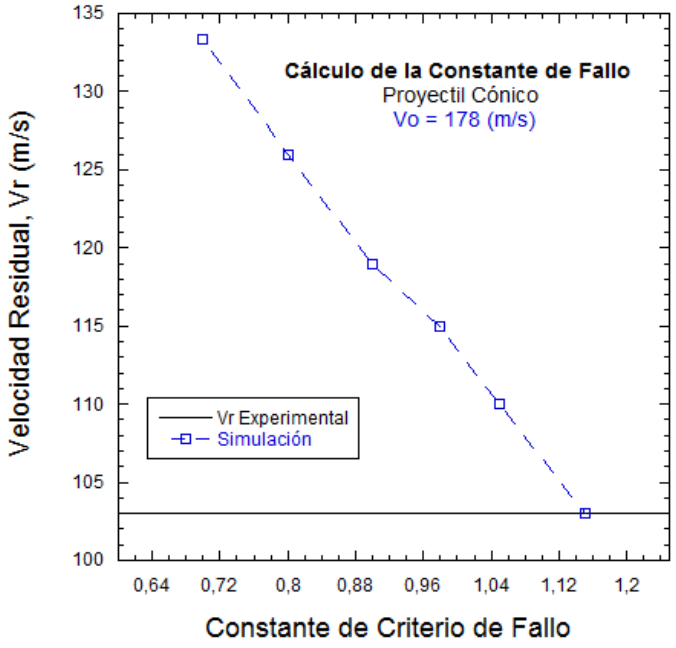


Gráfica 4. 2

Como se puede observar, la constante de fallo elegida para el proyectil plano, no coincide con el punto donde cortan. Esto se debe a que para el proyectil plano hubo que hacer este procedimiento a distintas velocidades, porque con V_0 = 178 m/s no era representativo del comportamiento a todas las velocidades.



Gráfica 4. 3



Gráfica 4. 4

Tanto para el proyectil hemisférico, como para el cónico, los resultados obtenidos para esta velocidad sí eran extrapolables para todo el rango de velocidades, de ahí que se cogiera como valor de Ɛ, el punto de corte de ambas gráficas.

Con lo obtenido en estas simulaciones se puede concluir que el valor de la constante de deformación de fallo para cada proyectil para la aleación de aluminio AA5757-H111, utilizando el método de Elementos Finitos Continuos Lagrangianos son las siguientes.

Proyectil	Cónico	Esférico	Plano
ε	1.15	1.02	0.52

Tabla 4. 4: Valores de la constante de fallo en función del material.

Una vez han sido calculadas las constantes para los tres modelos, se empezaron a realizar ensayos para el mismo rango de velocidades que se muestran en las perforaciones experimentales.

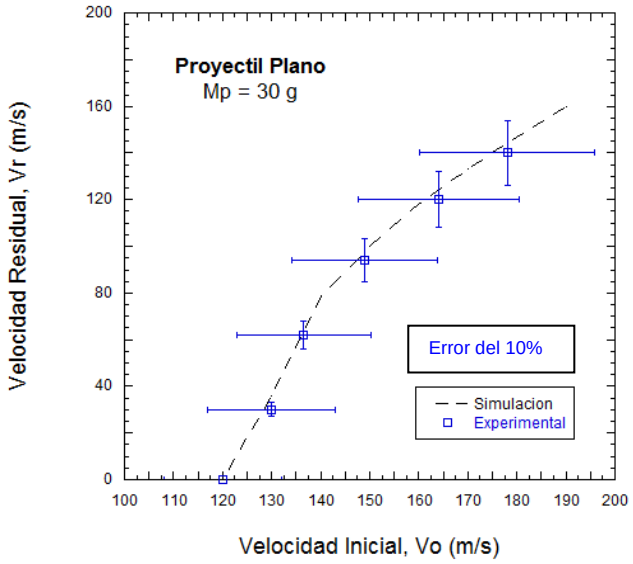
5. Comparación de los Resultados

Resumen del Capítulo

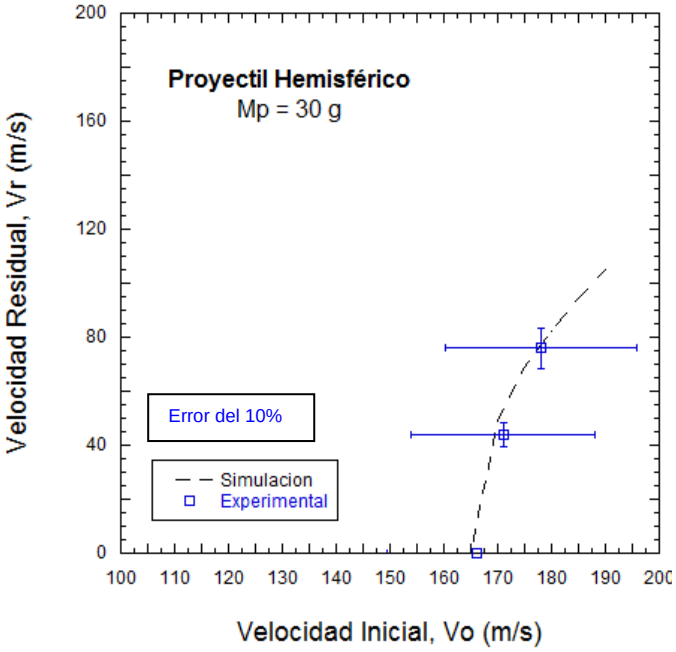
Con el diseño descrito en el capítulo anterior, se han realizado multitud de simulaciones con el fin de comparar los resultados con los valores experimentales y validar el modelo. En el estudio se comparan las velocidades residuales, la energía absorbida por la placa y la deformación.

5.1. Velocidades residuales

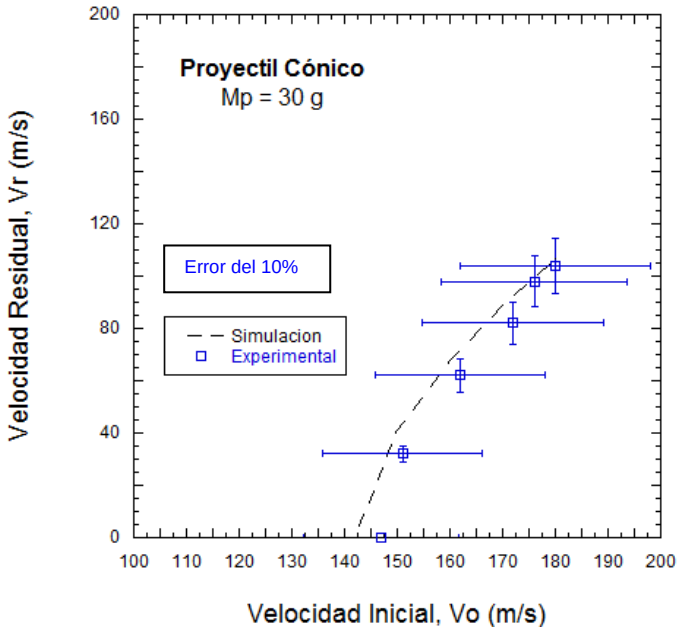
Las velocidades iniciales dependen de la geometría del proyectil, ya que en función de ésta la placa será perforada a más o menos velocidad. Como se puede observar en las gráficas adjuntadas, los valores experimentales tienen un error del 10% de su valor, tanto en la velocidad inicial (V_0), como en la velocidad residual del proyectil (V_r). Los resultados de las simulaciones están hechos con las constantes de criterio de fallo obtenidos, tal y como se describió en el capítulo anterior. A partir de dichos valores, se ha calculado una gráfica que facilita la velocidad residual del proyectil a partir de una velocidad inicial comprendida entre el límite balístico y 190 m/s.



Gráfica 5. 1



Gráfica 5. 2



Gráfica 5. 3

Como puede observarse en las gráficas, existe una velocidad inicial para la cual la velocidad residual tiene valor nulo. El valor inmediatamente anterior a esta velocidad de impacto, es lo que se conoce como límite balístico, que depende del material de la placa, el estado tensional y la velocidad de impacto.

Las gráficas muestran en los valores experimentales un margen de error del 10 % tanto en el eje X, como en el eje Y, debido a posibles errores en la medición.

Los resultados obtenidos en las simulaciones por ordenador son muy similares a los obtenidos en el laboratorio para todo el rango de velocidades estudiado, lo que es una importante muestra de que el modelo está bien diseñado.

5.2. Balance de Energías

La energía absorbida por la placa es otro baremo para comparar los modelos experimentales y de simulación. Como se explicó anteriormente, en los ensayos numéricos se ha calculado la energía absorbida, de manera independiente, por la parte de la placa que recibe directamente el impacto y por el resto de la misma, tal y como se muestra en las siguientes figuras.

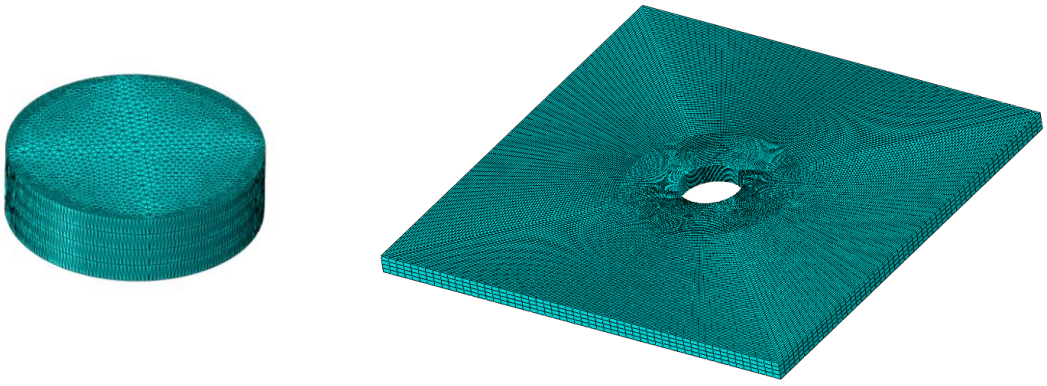


Figura 5. 1: Regiones estudiadas en el balance de energías

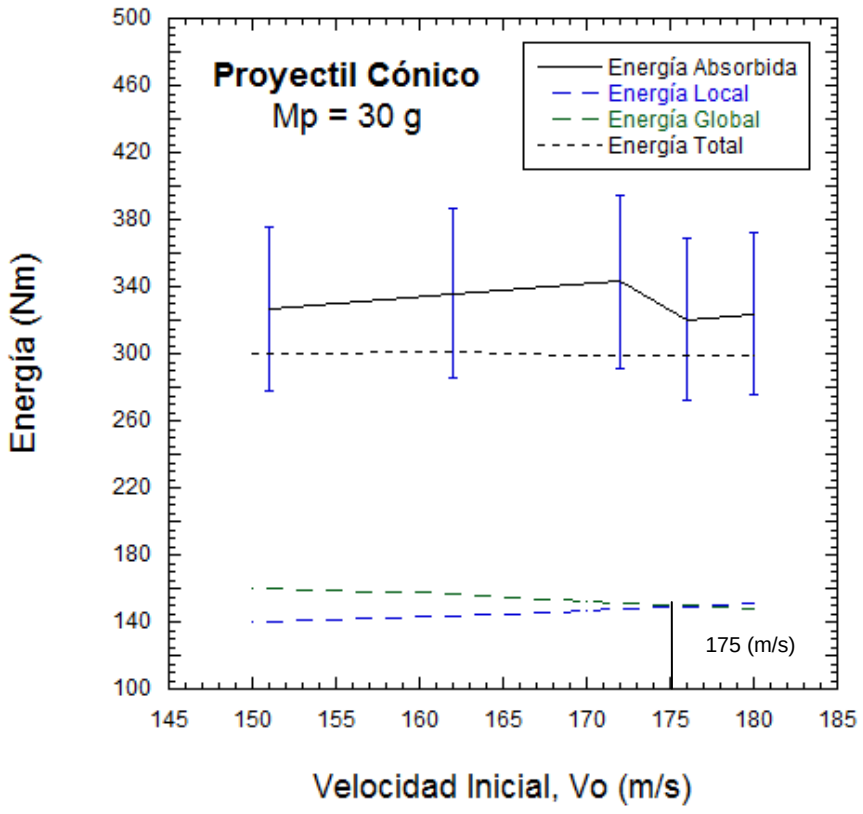
En cada una de las mostradas secciones se obtuvo tanto la energía cinética (con valores bastante pequeños) y la energía de total de deformación. Sumando dichos valores, se calcula la energía absorbida en la zona de impacto (Energía Local) y la absorbida en el resto de la placa (Energía Global). Si se suman ambas energías, se llega a la energía total absorbida en los cálculos numéricos.

A diferencia de las velocidades, los valores experimentales de la energía absorbida de pueden ser medidos de manera directa, por lo que se han de hacer una serie de cálculos para obtenerla. Tal y como se explica a continuación.

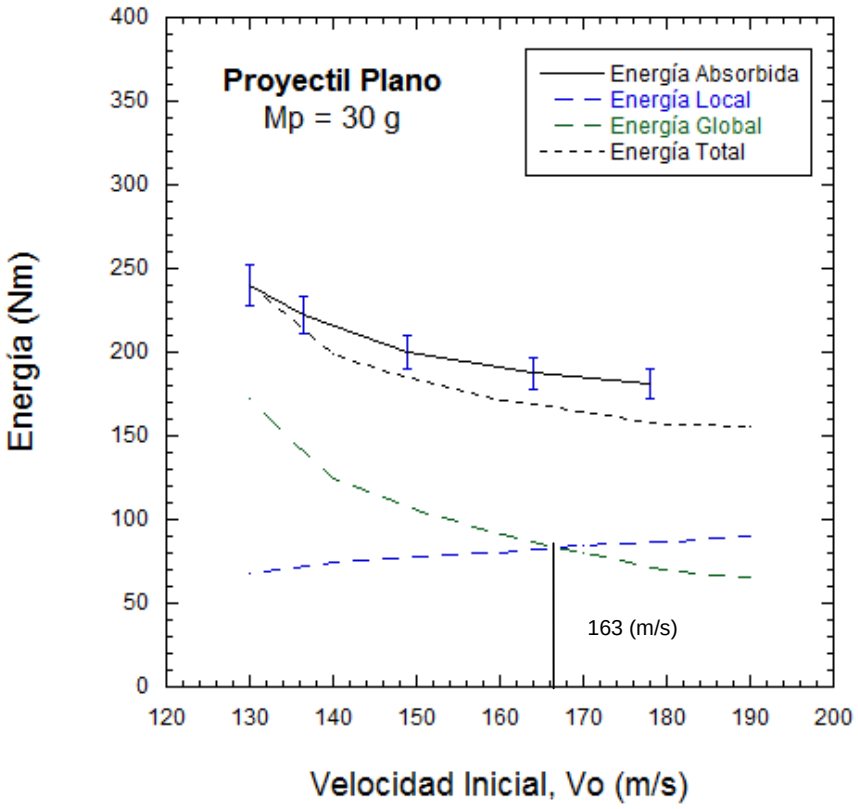
$$E_{Inicial} = E_{Final} \longrightarrow \tfrac{1}{2}\, m_{proyectil}\, V_0^2 \approx E_{Absorbida} + \tfrac{1}{2}\, m_{proyectil}\, V_r^2$$

$$E_{Absorbida} \approx \tfrac{1}{2}\, m_{proyectil}\, (V_0^2 - V_r^2) \qquad (3)$$

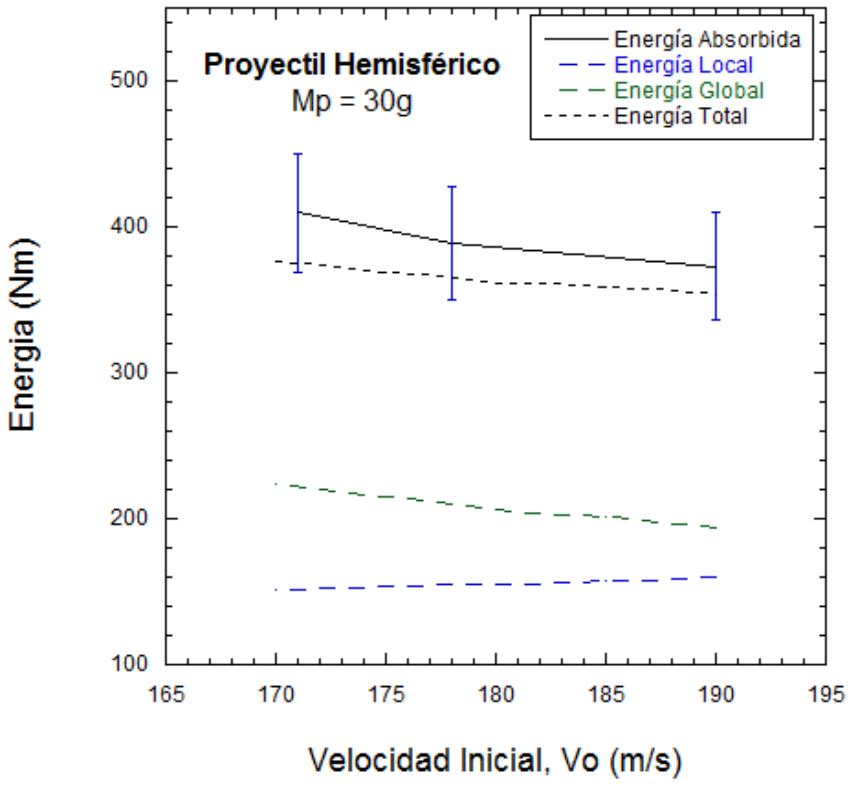
A partir de estos cálculos, tomando como V_0 y V_r los valores experimentales, obtenemos las energías absorbidas por la placa experimentalmente. Las cuatro energías que se han explicado, son las que se muestran en las figuras adyacentes para cada uno de los proyectiles.



Gráfica 5. 4



Gráfica 5. 5



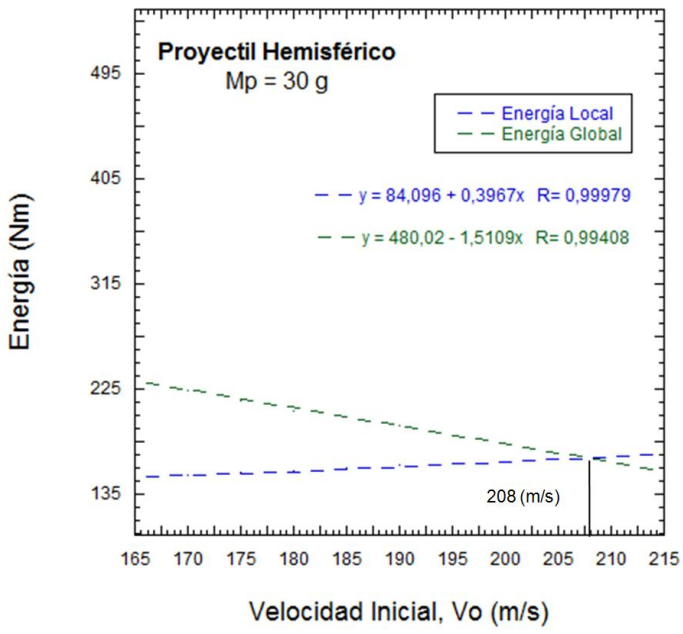
Gráfica 5. 6

Se puede ver como los resultados de experimentales están ligeramente por encima de los numéricos, pero éstos son bastante representativos y fiables, ya que muestran bastante bien la evolución de la energía en función de la velocidad, en comparación con los datos reales.

En las gráficas del proyectil cónico y plano puede observarse también que hay una determinada velocidad en la cual las energías local y global son iguales. Además, a velocidades bajas es menor la energía absorbida en la zona de impacto que en el resto de la placa. Sin embargo, a partir de una determinada velocidad V_0, que variará en función del proyectil, la energía local será mayor que la global, la cual puede llegar a ser casi nula par velocidades de impacto muy altas.

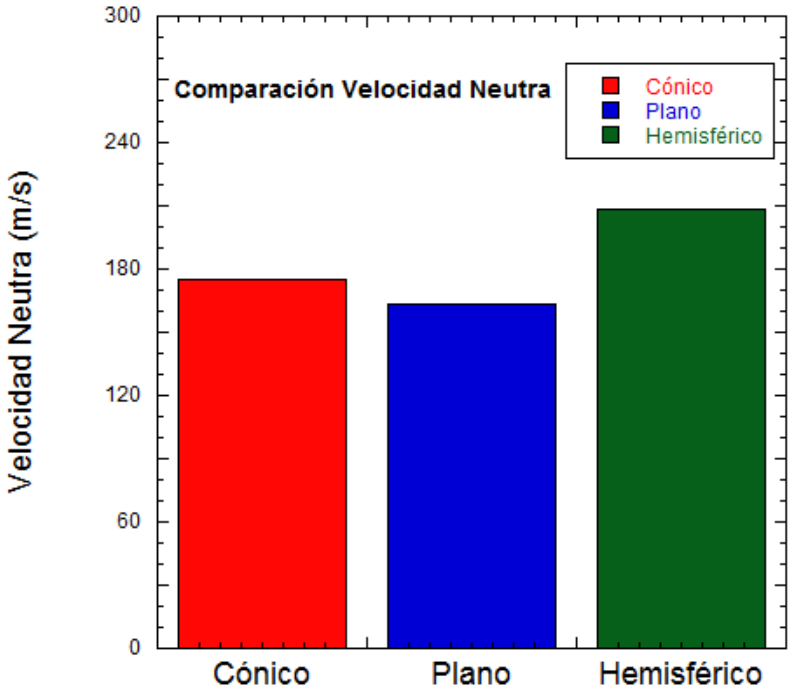
Esto implica que a bajas velocidades la parte de la placa que no que no recibe directamente el impacto, sufrirá mayores deflexiones por absorber mayor energía.

En el proyectil hemisférico, esta velocidad en la que se igualan las energías, está por encima del rango de velocidades de estudio. Así, al ver que el comportamiento de dichas energías en función de la velocidad sigue una regresión lineal, se han calculado las rectas y se han prolongado hasta el punto de corte, como se puede ver en la siguiente gráfica.



Gráfica 5. 7

Esta velocidad neutra sirve para valorar el rango de velocidades para el cual la placa sufre una deformación considerable, siendo mayor cuanto mayor sea la velocidad neutra. La gráfica 5.8 ilustra dichas velocidades en función de la geometría del proyectil.



Gráfica 5. 8

5.3. Perforación de la Placa

Cuando el proyectil perfora la aleación de aluminio, deja en la placa un orificio de diámetro similar al suyo. Esta perforación tendrá características diferentes según la velocidad y la geometría del proyectil, pudiéndose formar en los bordes una especie de pétalos, como se podrá ver en las figuras posteriores.

En este apartado se han hecho unas comparaciones gráficas entre el modelo numérico y los ensayos experimentales del aspecto de la perforación en cada uno de los proyectiles para una velocidad concreta.

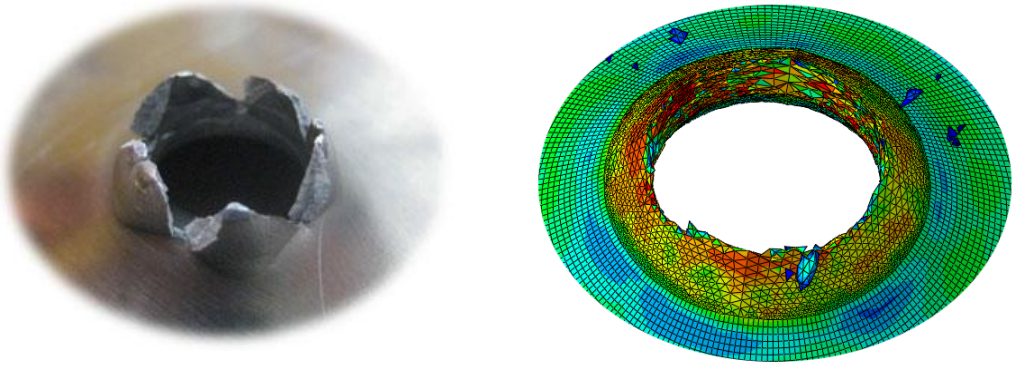


Figura 5. 2: Comparación de ensayos del proyectil cónico. V_0 = 179.86 m/s

El resultado del aspecto de la perforación en los dos ensayos no es tan parecido como podría esperarse. Esto se debe a que el criterio de fallo escogido en las simulaciones, ha sido aproximado a una simple constante que considera únicamente el estado tensional pero no es sensible a factores importantes como la temperatura, velocidad de deformación, triaxialidad y parámetro de Lode.

Sin embargo para las otras dos geometrías, las simulaciones sí que representan más fielmente los resultados obtenidos en el laboratorio, como puede apreciarse en las figuras 5.3 y 5.4

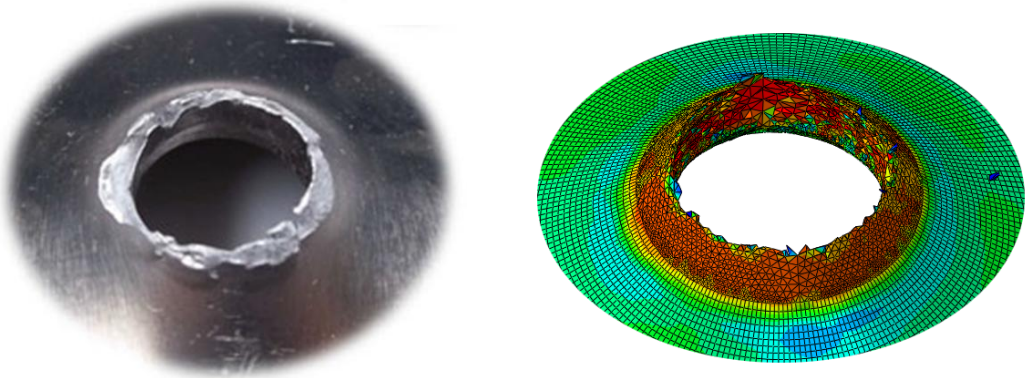


Figura 5. 3: Comparación de ensayos del proyectil hemisférico. V_0 = 166,67 m/s

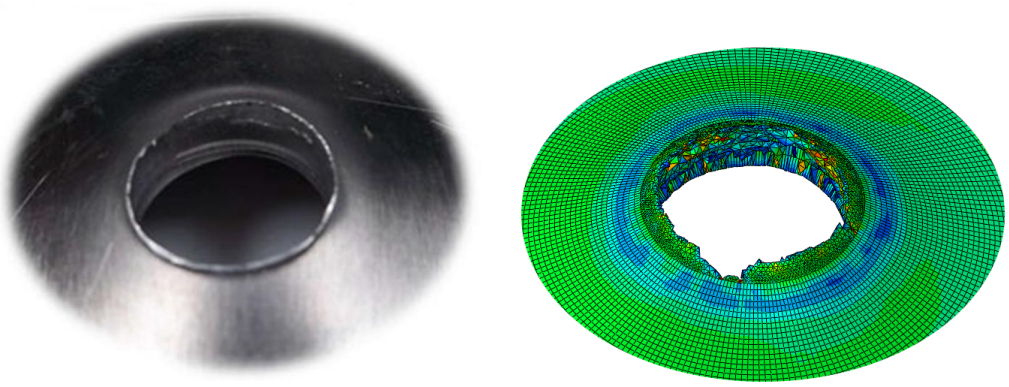


Figura 5. 4: Comparación de ensayos del proyectil Plano. V_0 = 136.6 m/s

5.4. Deflexión de la Placa

El impacto del proyectil contra la placa produce en ésta una deformación, más pronunciada en la parte central, que variará en función de la velocidad de impacto. Como se ha explicada en un epígrafe anterior, cuanto mayor sea esta velocidad menos se deformará la placa debido a que la energía global absorbida disminuye.

A continuación se muestran imágenes comparativas de los modelos experimentales y numéricos, para las tres geometrías a una determinada velocidad.

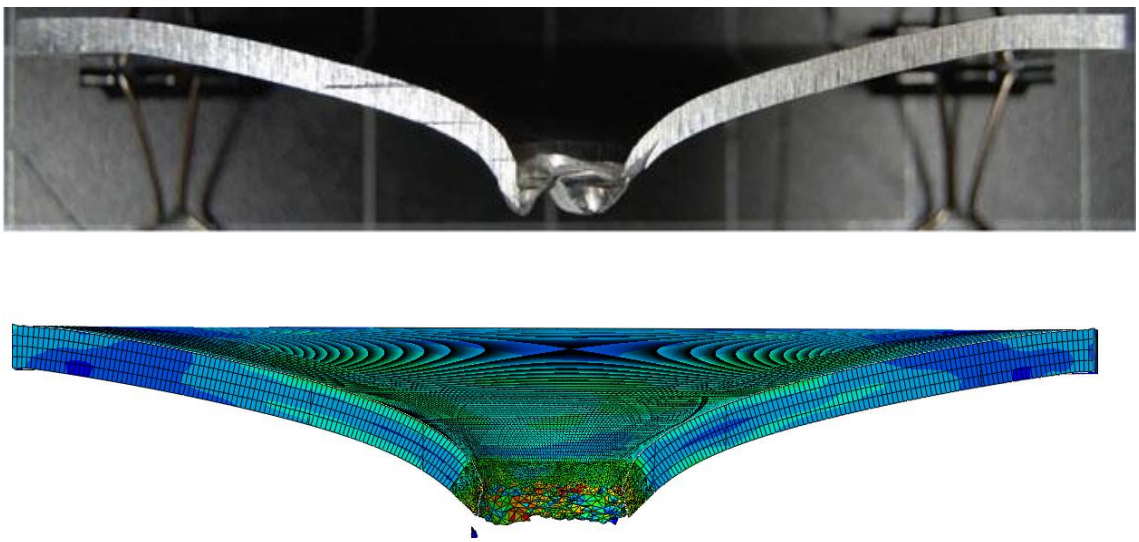


Figura 5. 5: Comparación de la deflexión en el proyectil cónico. V_0 = 143,68 m/s

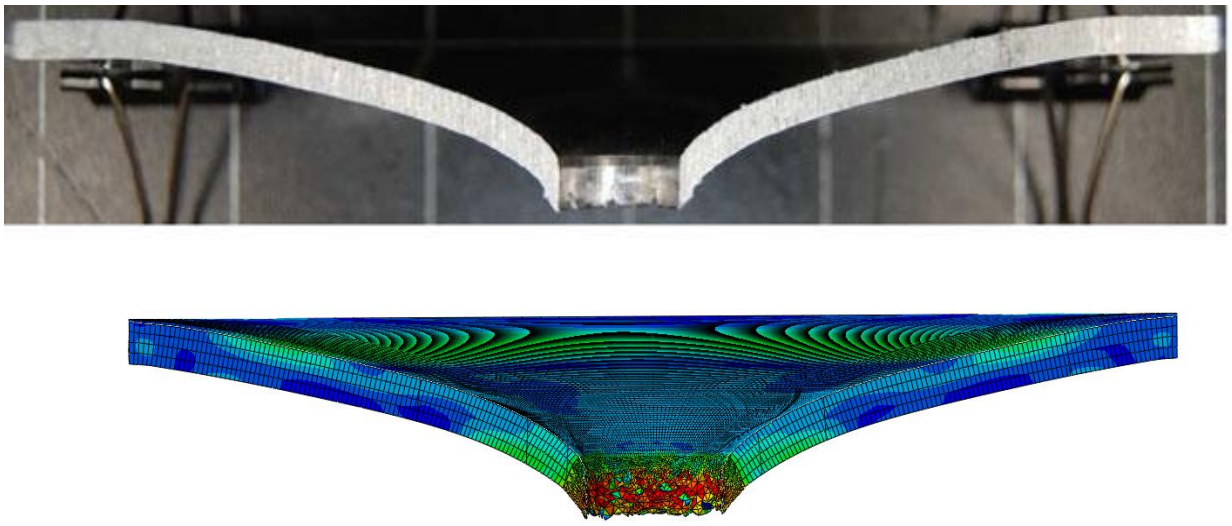


Figura 5. 6: Comparación de la deflexión en el proyectil hemisférico. V_0 = 178,57 m/s

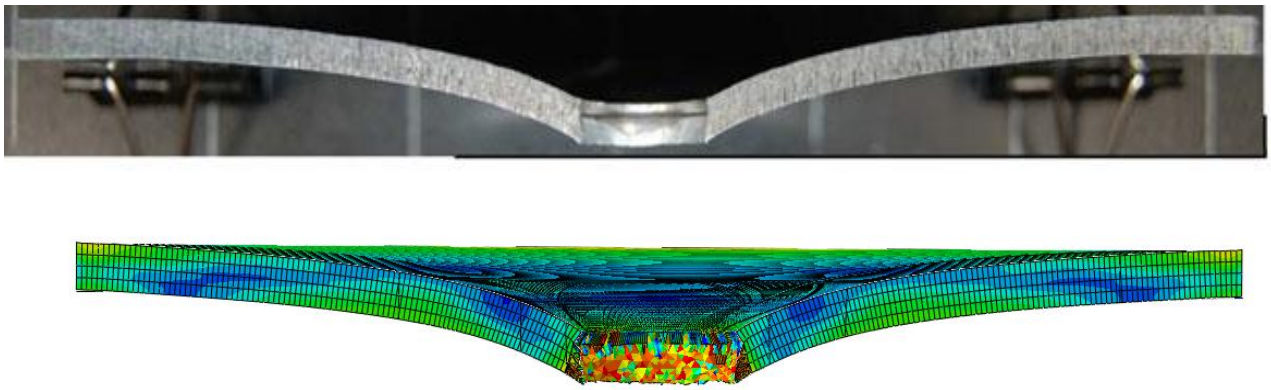


Figura 5. 7: Comparación de la deflexión en el proyectil plano. V_0 = 136,6 m/s

Las figuras 5.5, 5.6 y 5.7 muestran como las simulaciones numéricas reproducen de manera real el comportamiento a deflexión de la placa. Esto es una razón más para confirmar la validez del modelo de diseñado mediante ABAQUS Scripting.

6. Conclusiones

Respecto a los objetivos planteados durante la realización de este proyecto, se han determinado las siguientes conclusiones.

6.1. Comportamiento de placas de AA 5754-H111 sometidas a impactos con distintas geometrías de proyectil

Con la información mostrada en los capítulos anteriores se han alcanzado distintas conclusiones acerca del comportamiento del material.

- El criterio de fallo no puede aproximarse a una sola constante, ya que distintas magnitudes influyen sobre su comportamiento.
- La placa de AA 5754-H111 tiene un mayor límite balístico cuando recibe un impacto de un proyectil hemisférico, y el menor lo alcanza cuando es impactada por un proyectil plano.
- Se han encontrado diferencias en las energías local y global absorbidas por la placa en función del proyectil. Para velocidades iguales, en términos generales el proyectil hemisférico provoca una mayor energía total absorbida, seguido del cónico y después del plano.
- Además, la velocidad neutra sigue el mismo orden que las energías absorbidas. Siendo la mayor la del proyectil hemisférico, después el cónico y por último el plano. Esto supone que el proyectil hemisférico es el proyectil que provoca una deflexión considerable en un mayor rango de velocidades.
- En las simulaciones numéricas, se ha observado que mientras la energía local se mantiene prácticamente constante en todo el rango de velocidades, la energía global va disminuyendo según la velocidad inicial crece respecto al límite balístico.

6.2. Validación del modelo de elementos finitos

El principal objetivo del trabajo era diseñar un modelo de elementos finitos mediante ABAQUS Scripting, capaz de reproducir fielmente los valores alcanzados experimentalmente en el artículo “Experimental Study on the Perforation Process of 5754-H111 and 6082-T6 Aluminium Plates Subjected to Normal Impact by Conical, Hemisphericaland Blunt Project” realizado por realizado por M. Rodríguez-Millán, A. Vaz-Romero, A. Rusinek, J.A. Rodríguez-Martínez y A. Arias.

Como se ha demostrado en la comparación de resultados de ambos modelos, el modelo de simulación alcanza resultados muy similares a los publicados en el artículo citado, pero en mucho menos tiempo y de una manera sencilla.

La principal ventaja de este logro no reside solo en este trabajo en sí, si no en la facilidad con la que el modelo puede ser adaptado para cualquier modelo de impacto. Desde variantes en el material, como modificaciones en la geometría del proyectil, cambios en el rango de velocidades de estudio o incluso otros criterios de fallo.

En otras palabras, este modelo puede servir como plantilla para estudiar cualquier simulación de impacto que se considere oportuno, siendo una herramienta muy útil para posteriores investigaciones.

Anexo A: Creación de particiones y “Sets”

Particiones Circulares

```
#Create circular partitions

#Partition1

sen45 = 2 ** (-0.5)

f1 = shellPart.faces.findAt((DimensionPlaca/2.0, DimensionPlaca/2.0,
GrosorPlaca))

t1 = shellPart.MakeSketchTransform(sketchPlane=f1, origin=(DimensionPlaca/2,
DimensionPlaca/2, GrosorPlaca))

PartitionEdgeProfileSketch1 =
shellModel.ConstrainedSketch(name='PartitionEdgeProfileSketch1',sheetSize=5.0
, transform=t1)
PartitionEdgeProfileSketch1.CircleByCenterPerimeter(center=(0,0),
point1=(0.019*sen45,0.019*sen45))
PartitionEdgeProfileSketch1.setPrimaryObject(option=SUPERIMPOSE)

shellPart.projectReferencesOntoSketch(sketch=PartitionEdgeProfileSketch1,
filter=COPLANAR_EDGES)
shellPart.PartitionFaceBySketch(faces=f1, sketch=PartitionEdgeProfileSketch1)

partitionCells=shellPart.cells.findAt((DimensionPlaca/2, DimensionPlaca/2,
GrosorPlaca/2),)

e1 = shellPart.edges.findAt((DimensionPlaca/2, DimensionPlaca/2 + 0.019,
GrosorPlaca))
direction_edge_pt = (0,0,GrosorPlaca/2)
e2 = shellPart.edges.findAt((direction_edge_pt,))

shellPart.PartitionCellByExtrudeEdge(cells=partitionCells, edges=e1, line=
e2[0], sense=REVERSE)


#Partition2

f1 = shellPart.faces.findAt((DimensionPlaca/2.0, DimensionPlaca/2.0,
GrosorPlaca))

t1 = shellPart.MakeSketchTransform(sketchPlane=f1, origin=(DimensionPlaca/2,
DimensionPlaca/2, GrosorPlaca))

PartitionEdgeProfileSketch2 =
shellModel.ConstrainedSketch(name='PartitionEdgeProfileSketch2',sheetSize=5.0
, transform=t1)
PartitionEdgeProfileSketch2.CircleByCenterPerimeter(center=(0,0),
point1=(0.013*sen45,0.013*sen45))
PartitionEdgeProfileSketch2.setPrimaryObject(option=SUPERIMPOSE)

shellPart.projectReferencesOntoSketch(sketch=PartitionEdgeProfileSketch2,
filter=COPLANAR_EDGES)
shellPart.PartitionFaceBySketch(faces=f1, sketch=PartitionEdgeProfileSketch2)
```

```
partitionCells=shellPart.cells.findAt((DimensionPlaca/2, DimensionPlaca/2,
GrosorPlaca/2),)
e1 = shellPart.edges.findAt((DimensionPlaca/2, DimensionPlaca/2 + 0.013,
GrosorPlaca))
direction_edge_pt = (0,0,GrosorPlaca/2)
e2 = shellPart.edges.findAt((direction_edge_pt,))

shellPart.PartitionCellByExtrudeEdge(cells=partitionCells, edges=e1, line=
e2[0], sense=REVERSE)


#Partition3

f1 = shellPart.faces.findAt((DimensionPlaca/2.0, DimensionPlaca/2.0,
GrosorPlaca))

t1 = shellPart.MakeSketchTransform(sketchPlane=f1, origin=(DimensionPlaca/2,
DimensionPlaca/2, GrosorPlaca))

PartitionEdgeProfileSketch3 =
shellModel.ConstrainedSketch(name='PartitionEdgeProfileSketch3',sheetSize=5.0
, transform=t1)
PartitionEdgeProfileSketch3.CircleByCenterPerimeter(center=(0,0),
point1=(0.007*sen45,0.007*sen45))
PartitionEdgeProfileSketch3.setPrimaryObject(option=SUPERIMPOSE)

shellPart.projectReferencesOntoSketch(sketch=PartitionEdgeProfileSketch3,
filter=COPLANAR_EDGES)
shellPart.PartitionFaceBySketch(faces=f1, sketch=PartitionEdgeProfileSketch3)

partitionCells=shellPart.cells.findAt((DimensionPlaca/2, DimensionPlaca/2,
GrosorPlaca/2),)

e1 = shellPart.edges.findAt((DimensionPlaca/2, DimensionPlaca/2 + 0.007,
GrosorPlaca))
direction_edge_pt = (0,0,GrosorPlaca/2)
e2 = shellPart.edges.findAt((direction_edge_pt,))
```

"Sets"

```
#Create sets

#Create sets

set_cell1 = shellPart.cells.findAt(((DimensionPlaca/2,0,0),))
set_cell2 = shellPart.cells.findAt(((DimensionPlaca/2,DimensionPlaca/2,0),))
set_cell3 =
shellPart.cells.findAt(((DimensionPlaca/2+0.0071,DimensionPlaca/2,0),))
set_cell4 =
shellPart.cells.findAt(((DimensionPlaca/2+0.0131,DimensionPlaca/2,0),))

set_seq = set_cell1
set_seq +=set_cell4
set_seq +=set_cell3

Set1=shellPart.Set(name= 'Set-1', cells=set_cell2)
Set2=shellPart.Set(name= 'Set-2', cells=set_seq)

set_seq = set_cell2
set_seq +=set_cell3
Set3=shellPart.Set(name= 'Set-3', cells=set_seq)

set_seq = set_cell1
set_seq +=set_cell4
Set4=shellPart.Set(name= 'Set-4', cells=set_seq)

Set5=shellPart.Set(name='Set-5', cells=set_cell3)
```

Particiones de las diagonales

```
#Create Trianglar Partitions

shellPart.DatumPointByCoordinate(coords=(0,0,0))
shellPart.DatumPointByCoordinate(coords=(0,0,GrosorPlaca))
shellPart.DatumPointByCoordinate(coords=(0,DimensionPlaca,0))
shellPart.DatumPointByCoordinate(coords=(0,DimensionPlaca,0.004))
shellPart.DatumPointByCoordinate(coords=(DimensionPlaca,0,0))
shellPart.DatumPointByCoordinate(coords=(DimensionPlaca,DimensionPlaca,0))

shellPart_datums_keys = shellPart.datums.keys()
shellPart_datums_keys.sort()

shell_datum_point_1 = shellPart.datums[shellPart_datums_keys[0]]
shell_datum_point_2 = shellPart.datums[shellPart_datums_keys[1]]
shell_datum_point_3 = shellPart.datums[shellPart_datums_keys[2]]
shell_datum_point_4 = shellPart.datums[shellPart_datums_keys[3]]
shell_datum_point_5 = shellPart.datums[shellPart_datums_keys[4]]
shell_datum_point_6 = shellPart.datums[shellPart_datums_keys[5]]
```

```
partition_cell1 = shellPart.cells.findAt(((0,0,0),))
partition_cell2 = shellPart.cells.findAt(((DimensionPlaca/2-0.019/2-
0.013/2,DimensionPlaca/2,0),))
partition_cell3 = shellPart.cells.findAt(((DimensionPlaca/2-0.013/2-
0.007/2,DimensionPlaca/2,0),))
partition_cell4 = shellPart.cells.findAt(((DimensionPlaca/2,
DimensionPlaca/2,0),))

set_seq = partition_cell1
set_seq +=partition_cell2
set_seq +=partition_cell3
set_seq +=partition_cell4

shellPart.PartitionCellByPlaneThreePoints(point1=shell_datum_point_1,
point2=shell_datum_point_2, point3=shell_datum_point_6, cells=set_seq)

partition_cell1 = shellPart.cells.findAt(((0,DimensionPlaca,0),))
partition_cell2 = shellPart.cells.findAt(((DimensionPlaca/2-0.019/2-
0.013/2,DimensionPlaca/2,0),))
partition_cell3 = shellPart.cells.findAt(((DimensionPlaca/2-0.013/2-
0.007/2,DimensionPlaca/2,0),))
partition_cell4 =
shellPart.cells.findAt(((DimensionPlaca/2,DimensionPlaca/2,0),))
partition_cell5 =
shellPart.cells.findAt(((DimensionPlaca/2+0.007/2,DimensionPlaca/2,0),))
partition_cell6 =
shellPart.cells.findAt(((DimensionPlaca/2+0.013/2+0.007/2,DimensionPlaca/2,0)
,))
partition_cell7 =
shellPart.cells.findAt(((DimensionPlaca/2+0.019/2+0.013/2,DimensionPlaca/2,0)
,))
partition_cell8 = shellPart.cells.findAt(((DimensionPlaca,0,0),))

set_seq = partition_cell1
set_seq +=partition_cell2
set_seq +=partition_cell3
set_seq +=partition_cell4
set_seq +=partition_cell5
set_seq +=partition_cell6
set_seq +=partition_cell7
set_seq +=partition_cell8

shellPart.PartitionCellByPlaneThreePoints(point1=shell_datum_point_3,
point2=shell_datum_point_4, point3=shell_datum_point_5, cells=set_seq)
```

Anexo B: Definición del mallado

```
exteriorEdge1 = shellPart.edges.findAt((DimensionPlaca/2, 0, GrosorPlaca))
exteriorEdge2 = shellPart.edges.findAt((0, DimensionPlaca/2, GrosorPlaca))
exteriorEdge3 = shellPart.edges.findAt((DimensionPlaca, DimensionPlaca/2 ,
GrosorPlaca))
exteriorEdge4 = shellPart.edges.findAt((DimensionPlaca/2, DimensionPlaca,
GrosorPlaca))
shellPart.seedEdgeByNumber(edges=(exteriorEdge1,exteriorEdge2,exteriorEdge3,e
xteriorEdge4,), number=20)

triangularExteriorEdge1 = shellPart.edges.findAt((DimensionPlaca/2-0.019,
DimensionPlaca/2-0.019, GrosorPlaca))
triangularExteriorEdge2 = shellPart.edges.findAt((DimensionPlaca/2+0.019,
DimensionPlaca/2-0.019, GrosorPlaca))
triangularExteriorEdge3 = shellPart.edges.findAt((DimensionPlaca/2-0.019,
DimensionPlaca/2+0.019, GrosorPlaca))
triangularExteriorEdge4 = shellPart.edges.findAt((DimensionPlaca/2+0.019,
DimensionPlaca/2+0.019, GrosorPlaca))
shellPart.seedEdgeByNumber(edges=(triangularExteriorEdge1,triangularExteriorE
dge2,triangularExteriorEdge3,triangularExteriorEdge4,), number=15)

R3Edge1 = shellPart.edges.findAt((DimensionPlaca/2-0.019, DimensionPlaca/2,
GrosorPlaca))
R3Edge2 = shellPart.edges.findAt((DimensionPlaca/2+0.019, DimensionPlaca/2,
GrosorPlaca))
R3Edge3 = shellPart.edges.findAt((DimensionPlaca/2, DimensionPlaca/2-0.019,
GrosorPlaca))
R3Edge4 = shellPart.edges.findAt((DimensionPlaca/2, DimensionPlaca/2+0.019,
GrosorPlaca))
shellPart.seedEdgeByNumber(edges=(R3Edge1,R3Edge2,R3Edge3,R3Edge4,),
number=20)

triangularR3Edge1 = shellPart.edges.findAt((DimensionPlaca/2-0.013,
DimensionPlaca/2-0.013, GrosorPlaca))
triangularR3Edge2 = shellPart.edges.findAt((DimensionPlaca/2+0.013,
DimensionPlaca/2-0.013, GrosorPlaca))
triangularR3Edge3 = shellPart.edges.findAt((DimensionPlaca/2-0.013,
DimensionPlaca/2+0.013, GrosorPlaca))
triangularR3Edge4 = shellPart.edges.findAt((DimensionPlaca/2+0.013,
DimensionPlaca/2+0.013, GrosorPlaca))
shellPart.seedEdgeByNumber(edges=(triangularR3Edge1,triangularR3Edge2,triangu
larR3Edge3,triangularR3Edge4,), number=15)

R2Edge1 = shellPart.edges.findAt((DimensionPlaca/2-0.013, DimensionPlaca/2,
GrosorPlaca))
R2Edge2 = shellPart.edges.findAt((DimensionPlaca/2+0.013, DimensionPlaca/2,
GrosorPlaca))
R2Edge3 = shellPart.edges.findAt((DimensionPlaca/2, DimensionPlaca/2-0.013,
GrosorPlaca))
R2Edge4 = shellPart.edges.findAt((DimensionPlaca/2, DimensionPlaca/2+0.013,
GrosorPlaca))
shellPart.seedEdgeByNumber(edges=(R2Edge1,R2Edge2,R2Edge3,R2Edge4,),
number=20)
```

```
triangularR2Edge1 = shellPart.edges.findAt((DimensionPlaca/2-0.007,
DimensionPlaca/2-0.007, GrosorPlaca))
triangularR2Edge2 = shellPart.edges.findAt((DimensionPlaca/2+0.007,
DimensionPlaca/2-0.007, GrosorPlaca))
triangularR2Edge3 = shellPart.edges.findAt((DimensionPlaca/2-0.007,
DimensionPlaca/2+0.007, GrosorPlaca))
triangularR2Edge4 = shellPart.edges.findAt((DimensionPlaca/2+0.007,
DimensionPlaca/2+0.007, GrosorPlaca))
shellPart.seedEdgeByNumber(edges=(triangularR2Edge1,triangularR2Edge2,triangu
larR2Edge3,triangularR2Edge4,), number=15)

R1Edge1 = shellPart.edges.findAt((DimensionPlaca/2-0.007, DimensionPlaca/2,
GrosorPlaca))
R1Edge2 = shellPart.edges.findAt((DimensionPlaca/2+0.007, DimensionPlaca/2,
GrosorPlaca))
R1Edge3 = shellPart.edges.findAt((DimensionPlaca/2, DimensionPlaca/2-0.007,
GrosorPlaca))
R1Edge4 = shellPart.edges.findAt((DimensionPlaca/2, DimensionPlaca/2+0.007,
GrosorPlaca))
shellPart.seedEdgeByNumber(edges=(R1Edge1,R1Edge2,R1Edge3,R1Edge4,),
number=20)

triangularR1Edge1 = shellPart.edges.findAt((DimensionPlaca/2-0.007/2,
DimensionPlaca/2-0.007/2, GrosorPlaca))
triangularR1Edge2 = shellPart.edges.findAt((DimensionPlaca/2+0.007/2,
DimensionPlaca/2-0.007/2, GrosorPlaca))
triangularR1Edge3 = shellPart.edges.findAt((DimensionPlaca/2-0.007/2,
DimensionPlaca/2+0.007/2, GrosorPlaca))
triangularR1Edge4 = shellPart.edges.findAt((DimensionPlaca/2+0.007/2,
DimensionPlaca/2+0.007/2, GrosorPlaca))
shellPart.seedEdgeByNumber(edges=(triangularR1Edge1,triangularR1Edge2,triangu
larR1Edge3,triangularR1Edge4,), number=15)

thicknessEdge1 = shellPart.edges.findAt((0, 0, GrosorPlaca/2))
thicknessEdge2 = shellPart.edges.findAt((DimensionPlaca, 0, GrosorPlaca/2))
thicknessEdge3 = shellPart.edges.findAt((0, DimensionPlaca, GrosorPlaca/2))
thicknessEdge4 = shellPart.edges.findAt((DimensionPlaca, DimensionPlaca,
GrosorPlaca/2))
shellPart.seedEdgeByNumber(edges=(thicknessEdge1,thicknessEdge2,thicknessEdge
3,thicknessEdge4,), number=10)
```

7. Bibliografía

[1] International Aluminum Institute, Statistics, Map.

http://www.world-aluminium.org/statistics/#map

[2] J. Dwight. Aluminum design and construction. Technical report, University of Cambridge; and Fellow of Magdalene College, Cambridge, 1999.

[3] W. F. Smith. Ciencia e Ingeniería de Materiales. McGraw-Hill, 2004

[4] Asociación para el Reciclado de Productos de Aluminio, Usos y Propiedades del Aluminio.

http://aluminio.org/?p=821

[5] M. Rodríguez-Millán, A. Vaz-Romero, A. Rusinek, J.A. Rodríguez-Martínez and A. Arias “, Experimental Study on the Perforation Process of 5754-H111 and 6082-T6 Aluminium Plates Subjected to Normal Impact by Conical, Hemisphericaland Blunt Project”.

[6] Python Software Foundation, Documentation, Beginner’s Guide.

http://python.org/moin/BeginnersGuide/Overview

[7] G. Johnson and W. Cook, “Fracture characteristics of three metals subjected to various strains, strain rates, temperatures and pressures,” Engineering Fracture Mechanics, vol. 21, 1985.

[8] T. Borvik, M. Langseth, O. Hopperstad, and K. Malo, “Ballistic penetration of steel plates.,” International Journal of Impact Engineering, 1999.

[9] A. Rusinek, J. Rodriguez-Martinez, A. Arias, J. Klepaczko, and J. Lopez-Puente, “Influence of conical projectile diameter on perpendicular impact of thin steel plate,” Engineering FractureMechanics, July 2008

[10] T.Wierzbicki, Y. Bao, Y.-W.Lee and Y.Bai,Calibration and evaluation of seven fracture models,"International Journal of Mechanical Sciences, vol. 47, pp. 719–743, Apr. 2005.

[11] G. Puri, Python scripts for Abaqus. Learn by example, United States of America, 2011.

[12] *Abaqus Scripting Reference Manual,* ABAQUS 6.12.

[13] *Abaqus Analysis User's Manual,* ABAQUS 6.12.

UNIVERSIDAD CARLOS III DE MADRID

Departamento de Mecánica de Medios Continuos y Teoría de Estructuras

Grado en Ingeniería Tecnologías Industriales

Trabajo Fin de Grado

Análisis del Fallo en Componentes Estructurales Metálicos Sometidos a Procesos de Perforación

Alumno:
Víctor Antona García

Tutor:
Marcos Rodríguez MIllán

Julio 2014

Página en blanco intencionadamente

1. Introducción

Junto al PDF de la memoria se adjunta el archivo .py que se ha desarrollado para realizar las simulaciones por ordenador en Python Abaqus Scripting.

Como se ha comentado en la memoria, este código puede servir como plantilla para la realización de cálculos numéricos de distintos ensayos de perforación, por lo que se ha considerado importante adjuntarlo.

2. Instrucciones del archivo py.

Para ver este archivo solo hay que abrirlo con el programa Notepad ++

Para utilizar el código en Abaqus, se puede usar abriendo Abaqus/CAE y seleccionar la opción “Run Script” y seleccionar el archivo.

Después, habrá que ir respondiendo a las preguntas que va haciendo Abaqus. Los valores utilizados en estos ensayos están escritos en el código debajo de la sentencia de cada pregunta cómo se explica en la memoria.